VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Inteligentní měřící a zobrazovací moduly s ESP8266

Intelligent Measurement and Visualization Modules with ESP8266

2018 Bc. Tanasis Vlachopulos

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tanasis Vlachopulos**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Inteligentní měřící a zobrazovací moduly s ESP8266
Intelligent Measurement and Visualization Modules with ESP8266

Jazyk vypracování: čeština

Zásady pro vypracování:

ESP8266 je levný WiFi System on Chip modul (SoC) používaný jako bezdrátový převodník pro sériové rozhraní. Použitý SoC má plně integrovaný TCP/IP stack a umožňuje jednoduché programování pomocí skriptovacího jazyku LUA nebo C++. Cílem diplomové práce je vytvořit ucelený systém tvořený těmito moduly a připojit je k vhodným senzorům a zobrazovacím prvkům. Součástí řešení bude server, zprostředkující vzájemnou komunikaci všech zapojených modulů. Navržené řešení má umožnit komunikaci libovolného počtu modulů a PUSH notifikace s možností dynamického přidávání modulů do sítě.

1. Seznamte se s problematikou IoT a používanými technologiemi.
2. Popište možnosti obvodu ESP8266 a dostupných SDK.
3. Navrhněte a podrobně popište řešení inteligentních modulů s ESP8266.
4. Otestujte navržené řešení s různými periferiemi a parametry sítě.
5. Zhodnoťte dosažené výsledky a porovnejte je s obdobnými projekty.

Seznam doporučené odborné literatury:

[1] Marco Schwartz, Internet of Things with ESP8266, Packt Publishing, 2016, ISBN 978-1786468024
[2] Alasdair Allan, Learning Esp8266: Build the Internet of Things with the Arduino Ide and Raspberry Pi, O'Reilly, 2017, ISBN 978-1491964279
[3] Lucy Rogers, Andy Stanford-Clark, Wiring the IoT: Connecting Hardware with Raspberry Pi, Node-RED, and MQTT, O'Reilly, 2017, ISBN 978-1491953334

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018

.....

Rád bych poděkoval mému vedoucímu práce, panu Mgr. Ing. Michalu Krumniklovi, Ph.D., za velmi cenné rady a doporučení a také za čas strávený při konzultacích, čehož si velice cením.

Abstrakt

Cílem této práce je seznámení čtenáře s problematikou IoT, takzvaného internetu věcí, a běžně používanými technologiemi, zejména pak s mikropočítačovými moduly ESP8266 a ESP32.

V úvodní části této práce jsem analyzoval existující IoT systémy a definoval seznam požadovaných funkcionalit. Dále jsem charakterizoval mikropočítačové moduly ESP8266 a ESP32. Zhodnotil jsem také různé nástroje k vývoji aplikací pro tyto moduly.

Dále jsem navrhl a implementoval systém, který umožňuje komunikaci s moduly, jejich správu, archivaci dat a vizualizaci dat. Součástí práce je server, který zajišťuje komunikaci prostřednictvím protokolu MQTT.

Připojené moduly mohou sloužit jak pro sběr dat ze senzorů, tak pro vizualizaci dat. Pro moduly jsem vytvořil klientskou knihovnu, pomocí vývojových nástrojů pro platformu Arduino, která usnadňuje komunikaci se serverem.

Klíčová slova: IoT, domácí automatizace, ESP8266, ESP32, MQTT, Arduino, Python, WiFi

Abstract

The aim of this work is introducing the reader to IoT, also known as the Internet of Things, and commonly used technologies in this area, especially with the microcomputer modules ESP8266 and ESP32.

In the theoretical part of this work, I analyzed the existing IoT systems and defined the list of required functionalities. I also described the ESP8266 and ESP32 microcomputer modules and reviewed various application development tools.

In this thesis, I designed and implemented a system that allows communication with modules, their management, data archiving and data visualization. Part of the solution is the server application that provides communication via MQTT protocol.

Connected modules can be used as sensors or for data visualization. For the modules I create a client library, using development tools for Arduino platform, which enables communication with the server application.

Key Words: IoT, home automation, ESP8266, ESP32, MQTT, Arduino, Python, WiFi

Obsah

Seznam použitých zkratek a symbolů

ACL	–	Access Control List
DBMS	–	Database Management System
IDE	–	Integrated Development Environment
IoT	–	Internet of Things
ORM	–	Objektově Relační Mapování
OTA	–	Over The Air programming
RSSI	–	Received Signal Strength Indication
SDK	–	Software Development Kit
TLS	–	Transport Layer Security
WYSIWYG	–	What you see is what you get

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Když se na počátku tohoto tisíciletí poprvé objevil termín Internet věcí (the Internet of Things), byla tato představa pro většinu z nás spíše hudbou vzdálené budoucnosti než realitou a to zejména kvůli chybějícím a drahým technologiím. Internet věcí v dnešní době chápeme, jako rozsáhlou síť senzorů a zařízení, která sbírají data, popřípadě ovládají nějaká jiná zařízení, do jisté míry jsou zařízení schopny operovat autonomně, ale svá data mohou distribuovat skrze telekomunikační infrastrukturu a mohou být také vzdáleně řízena. Do této kategorie můžeme zahrnout širokou škálu zařízení od jednoduchých senzorů sbírající data, přes chytré zásuvky, žárovky, GPS sledovače, kamery, až po kybernetické implantáty a biosenzory.

Průmysl myšlenku Internetu věcí adaptoval poměrně rychle, chytrá řešení dokázala šetřit čas i peníze. Případů užití je nesčetně, ukázkovým příkladem využití těchto technologií jsou například projekty: sledování rozsáhlých ploch, jako jsou parkoviště, vinice a výrobní haly, koncept Smart Cities, nebo aplikace v oblasti automotive. Pro nasazení v domácím prostředí zde ovšem existovala zásadní překážka a tou byla cena zařízení, která se pohybovala v řádu stovek až tisíců korun, proto bylo nákladné vytvořit komplexní systém vzájemně komunikujících zařízení.

Významnou roli na poli domácí automatizace sehrál projekt Arduino, který vznikl v roce 2005 a jeho cílem bylo vytvořit jednoduchou prototypovací platformu pro studenty [1]. Projekt byl velmi úspěšný, vývojové desky byly cenově dostupné a pro vývoj programů bylo připraveno jednoduché IDE s podporou jazyka C++ a nadstavbou Wiring. Vývojové kity se tak staly velmi populární v komunitě kutilů i mezi uživateli bez hlubokých technických znalostí. Postupem času se objevila na trhu celá řada klonů Arduina s cenou v řádu několika dolarů.

Nedostatkem většiny Arduino desek byla absence bezdrátové konektivity, tento problém se obvykle řešil připojením různých rozšiřujících modulů, který zprostředkovaly konektivitu přes WiFi, Bluetooth, pásmo 433 MHz a jiné. Jedním z výrobců, který začal vyrábět rozšiřující moduly bezdrátové konektivity byl Espressif System, ten v roce 2014 představil WiFi modul ESP8266.

Výkonný a levný modul ESP8266 mohl pracovat nejen jako rozšiřující WiFi-sériový převodník pro Arduino, či jiný vývojový modul, ale mohl i sám o sobě sloužit, jako programovatelný multifunkční vývojový modul. Díky ceně okolo 2 dolarů a částečné kompatibilitě s vývojovými nástroji pro Arduino se stal velmi populární alternativou k Arduinu a eskaloval tak zájem o domácí automatizaci a IoT. Na základě úspěchu ESP8266 představil Espressif System v roce 2016 nástupce ESP32, u kterého byly odstraněny mnohé nedostatky předešlého modelu a byla přidána široká škála nových funkcionalit, kromě WiFi konektivity disponuje ESP32 také technologií Bluetooth.

V této práci se budu zabývat představením několika existujících řešení pro integraci modulů ESP8266 a ESP32 v ucelený IoT systém. Shrnu nejpodstatnější vlastnosti a nedostatky existujících projektů a pokusím se navrhnout vlastní systém, pro odesílání a sběr dat ze zařízení ESP a jejich následnou prezentaci uživateli.

2 Systémy domácí automatizace

Pro použitelný IoT systém je pochopitelně potřeba více než jen samotné moduly, je nutné zajistit jejich připojení do sítě, správu, výměnu dat mezi zařízeními, sběr dat ze zařízení a ovládání zařízení, pro tyto účely vznikla řada IoT systémů, nebo jinými slovy systémů pro domácí automatizaci. V následující kapitole představím nejpoužívanější a nejzajímavější z nich. Existuje velké množství různých volně dostupných i komerčních systému specializovaných na různá zařízení, já se v této práci zaměřím především na ty svobodně použitelné, které umožňují správu libovolných zařízení uživatele, popřípadě zařízení ESP. Komerční systémy, jejichž součástí je řídící software a proprietární moduly, nejsou v této práci zahrnuty.

2.1 Domoticz

Domoticz je jeden z největších a nejstarších systémů pro domácí automatizaci. Vznikl v roce 2012, jako „domácí“ projekt a je stále vyvíjen a udržován. Domoticz funguje jako server, který zprostředkovává uživateli webové rozhraní, přes které lze přidávat zařízení, upravovat nastavení serveru, ale lze také vytvořit uživatelem definované dashboardy, uživatel si tedy může zvolit které zařízení a informace se mu zobrazí na titulní straně.

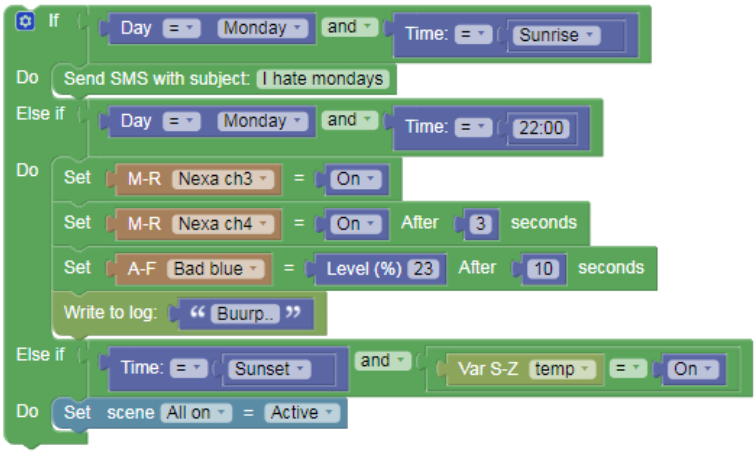
Server lze provozovat na různých platformách, jsou podporovány operační systémy Windows, Linux, macOS, ale i například NAS úložiště. Pro uživatele jsou připraveny i klientské aplikace pro Android a iOS, určené pro vzdálené ovládání a nahlížení.

Systém je napsán v jazyce C++ a používá vlastní implementaci webového serveru, uživatel ovšem může rozšiřovat funkcionalitu pomocí skriptů. Primárním skriptovacím jazykem je Lua, podpora tohoto jazyka je zabudovaná pro všechny dostupné platformy. Alternativně může uživatel použít i jiné skriptovací jazyky jako Bash, Python, Perl a podobné, při vytváření musí pouze nastavit cestu ke správnému interpretu.

Zajímavou funkcionalitou je rozhodně podpora vizuálního frameworku Blockly, který umožňuje definovat skripty pomocí grafických bloků. Na obrázku 1 je vidět jednoduché chování definované promocí Blockly. Blockly zpřístupňuje tvorbu uživatelsky definovaných automatizačních scénářů i uživatelům s malými znalostmi programování.

Domoticz se snaží být uživatelsky přívětivé i pro uživatele bez hlubokých technických znalostí a proto předdefinované profily pro zařízení, které je možno k serveru připojit. List je obsáhlý a zahrnuje téměř všechny na trhu dostupné zařízení [7]. Připojit lze samozřejmě i zařízení které nejsou zahrnuta do seznamu podporovaného hardwaru, definice vlastního zařízení je ovšem trochu složitější.

Zařízení s Domoticz serverem mohou komunikovat různými způsoby, nejobvyklejší metodou je ovšem HTTP API, v takovém případě posílají zařízení na server GET požadavky s daty, které chtějí předat. Podporované jsou také MQTT, Bluetooth, 433 MHz, ale i některé poněkud exotické protokoly jako Apple Homekit, eHouse, nebo Z-wave. Úplný seznam podporovaných protokolů je k dispozici na webových stránkách [8].



Obrázek 1: Skript vytvořený pomocí vizuálního frameworku Blockly (převzato z [6])

2.2 ESPEasy - Let's Control It

Systém ESPEasy na rozdíl od většiny ostatních systémů nefunguje na principu klient-server, celá logika automatizačního sytému se odehrává na zařízení ESP, ESPEasy je tedy firmwarem, který uživatel nahraje na své zařízení. K nahrání firmwaru na zařízení lze použít jednoduchý nástroj, který připravili vývojáři projektu. Firmware ESPEasy v současnosti podporuje pouze zařízení ESP8266, podpora novějších ESP32 je plánována. Po úspěšném nahrání vytvoří ESPEasy na zařízení webový server, pomocí kterého může uživatel zařízení konfigurovat. Především může nastavit parametry připojení k síti a způsob jakým může svá data předávat jiným systémům, které budou data zpracovávat.

Součástí webového rozhraní je nabídka, která umožní vybrat z předem definovaných senzorů, uživatel pouze vybere typ senzoru a definuje vstupy ke kterým bude senzor připojen. ESPEasy podporuje přibližně 55 nejběžnějších typů senzorů a výstupních zařízení, exotičtější hardware ovšem chybí, možnosti konfigurace daných senzorů jsou také omezené. Přestože ESPEasy umožňuje data předávat dalším systémům, není to nutné, ve webovém rozhraní zařízení je možné naměřená data zároveň i zobrazit.

Naměřená data lze ze zařízení odesílat do dalších IoT systémů, předdefinovány jsou Domoticz, OpenHAB, PiDome, Nodo, ThingSpeak, EmonCMS, databáze InfluxDB, ale lze použít i generické MQTT a HTTP.

2.3 Blynk

Jedná se o částečně komerční automatizační řešení založené cloudové infrastruktuře, uživatel si do mobilního telefonu stáhne aplikaci Blynk, zaregistruje se a vytvoří v aplikaci nový projekt. Společně s novým projektem se vygeneruje i takzvaný autentizační token, který uživatel později použije pro spárování se zařízením. V mobilní aplikaci uživatel zvolí své zařízení, podporované

jsou ESP8266, ESP32, většina Arduino modulů a jejich klonů[1], moduly Energia, platforma Particle a moduly poháněné Luou, Pythonem, nebo JavaScriptem. Součástí jednoho projektu může být i více zařízení. Uživatel následně dostává k dispozici jakousi virtuální pracovní deska, na kterou může rozmisťovat widgety, ty reprezentují jednotlivé funkce modulu, například tlačítko, přepínač, LED diodu, displej, ale i pokročilé chování jako například notifikace a jiné reakce na události. Uživatel může zdarma vytvářet pouze projekty s omezenou komplexitou, přidávání většího množství widgetů je zpoplatněno.

Pro dané zařízení je ještě nutné vytvořit kód, který implementuje námi požadované chování a předat mu autentizační token. Blynk poskytuje knihovny pro C++, které umožní spárování fyzických periferií s virtuálními widgety. Uživatel například vytvoří widget tlačítka, které bude ovládat LED diodu připojenou k zařízení, v C++ následně deklaruje událost „stisk tlačítka“, ve které definuje chování, tedy odeslání úrovně `HIGH/LOW` na příslušný výstup zařízení.

Komunikaci mezi zařízením a aplikací zajišťuje ve výchozím nastavení cloudový server provozovatele Blynk, uživatel má ovšem možnost si tento server zprovoznit i lokálně ve své síti. Komunikace je realizovaná pomocí protokolu HTTP, ve fázi testování je ovšem i podpora protokolu MQTT.

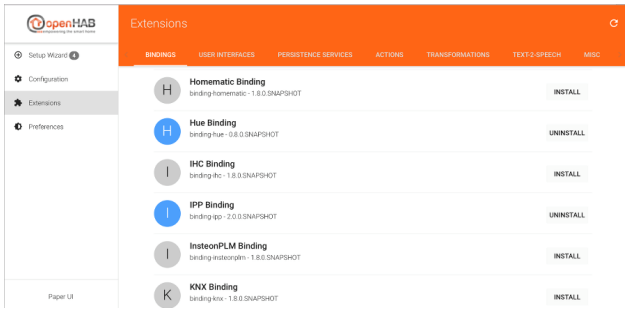
2.4 OpenHAB

OpenHAB je velmi rozsáhlé a komplexní řešení domácí automatizace, neslouží pouze pro sběr a zobrazení dat, ale může propojovat široké spektrum zařízení od senzorů, chytrých zásuvek, zahradních postřikovačů až po multimediální centra, televize a kamery. Tyto napojené systémy jsou nazývány „bindings“ a jsou distribuovány v podobě přídavných modulů, uživatel tedy dodatečně nainstaluje přídavné moduly pro systémy které chce používat. OpenHAB se poté, v mnoha případech, pokusí automaticky zařízení nalézt v lokální síti a navázat s ním komunikaci. Podobně jako Domoticz si jej uživatel stáhne a spustí na svém serveru, řešení je vytvořené v jazyce Java, je tedy multiplatformní. Na obrázku 2 je ukázka webového uživatelského rozhraní s nabídkou pro instalaci přípojných modulů. K dispozici je také mobilní aplikace, kterou může uživatel používat pro ovládání zařízení.

Po spárování se zařízeními si v prostředí nazvaném HABPanel uživatel vytváří různé druhy ovládacích panelů s widgety pro zobrazení dat a ovládání zařízení. Ukázka ovládacího panelu s ovládáním světel, topení a zobrazením aktuální teploty, je na obrázku 3. Vytvářet lze pochopitelně i vlastní napojení, pomocí jazyku Java, nebo uživatelské widgety v jazyce HTML.

Podobně jako bindings lze doinstalovat i různé reakční moduly, ty jsou nazývány jako „actions“, lze tak zpřístupnit odesílání e-mailů, MQTT zpráv, Twitterových zpráv a podobných, jako reakci na akce zařízení. Systém lze také rozšířit o různé napojení na databáze, například InfluxDB, SQLite, MySQL, atd., lze jej rozšířit o dodatečné moduly uživatelského rozhraní, kupříkladu různé stavové panely, grafické vizualizační nástroje a v neposlední řadě lze přidat i moduly pro převod

[1]Pokud nemá zařízení ve výchozím stavu hardware potřebný pro připojení k internetu, je nutné zařízení doplnit o nějaký bezdrátový, nebo ethernetový shield.



Obrázek 2: Zařízení v systému OpenHAB [9].

Obrázek 3: Ovládací panel HABPanel [9].

textu na řeč. Pro pokročilou automatizaci lze vytvářet i pravidla ve smyslu „when... then...“, uživatel tak může vytvořit například scénáře, kdy při otevření oken dojde k vypnutí vytápění v místnosti a aktivaci světel.

2.5 Home Assistant

Řešení Home Assistant, dále jen Hass, je z pohledu funkcionality podobné systému OpenHAB, technické provedení je ovšem odlišné. Hass obdobně jako jiné systémy funguje, jako serverová aplikace s webovým rozhraním v domácí síti uživatele. Pro instalaci je k dispozici několik metod. Systém Hass byl vyvinut v programovacím jazyce Python, lze jej tedy stáhnout a nainstalovat pomocí balíčkovacího systému PIP, který slouží právě pro distribuci různorodých knihoven v jazyce Python. Pro méně zkušené uživatele je připraven obraz operačního systému pro počítač RaspberryPi, tento operační systém vývojáři nazývají Hassbian. Hassbian přináší nejen snadnější instalaci, ale přináší i několik bonusových funkcí spojených s přímou kontrolou hardwaru počítače. Tvůrci toto spojení počítače RaspberryPi a systému Hass označují názvem Hass.io, z počítače se tak dle jejich slov stává „ultimate home automation hub“ [10].

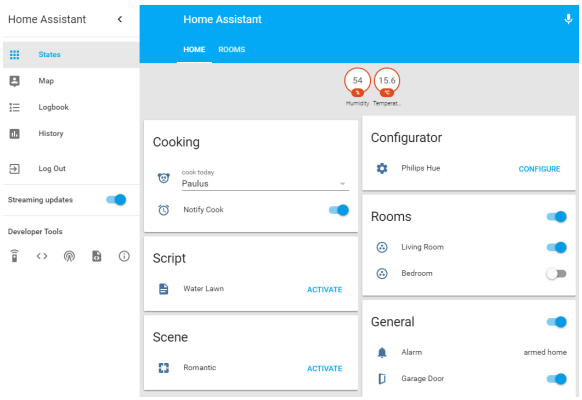
Chování systému uživatel definuje pomocí konfiguračních souborů formátu YAML, ve kterých specifikuje formát příchozích dat a způsob jakým budou uživateli prezentována ve webovém rozhraní. Na obrázku 4 je konfigurační soubor s nastavením pro připojení k MQTT brokeru a definicí jednoho senzoru, odesílajícího data pomocí protokolu MQTT. Hass, ve výchozím stavu, komunikuje se zařízeními pomocí protokolu MQTT, pro tyto účely má integrovaný MQTT broker, ale je možné používat i externí MQTT broker. Konfiguračními soubory je možné definovat podmínky a spouštěče, uživatel tak může vytvořit automatizační scénář, kdy akce na jednom zařízení vyvolá určitou reakci na zařízení jiném, nebo dojde k odeslání upozornění. Pro pokročilejší automatizaci je možné definovat vlastní skripty v jazyce Python.

Pokud základní funkcionalita systému uživateli nepostačuje, může jej rozšířit o celou řadu komponent, těch je v současnosti k dispozici více než tisíc [11]. Jedná se o konektory na různé existující IoT řešení, napojení na multimediální systémy, webové služby, monitorovací protokoly, nebo také integrace s hlasovými asistenty a službou IFTTT [13]. Pokud uživatel používá distribuci Hassbian může navíc připojit zařízení přes GPIO vstupy počítače RaspberryPi, využívat

```yaml
mqtt:
  broker: 192.168.1.1
  username: user
  password: passwd

sensor:
  platform: mqtt
  name: "Temperature"
  state_topic: "sensor/temperature"
  qos: 0
  unit_of_measurement: "ºC"
```

Obrázek 4: Konfigurace zařízení v jazyce YAML [14].



Obrázek 5: Ukázkový ovládací panel.

integrovaného Bluetooth rozhraní, nebo doinstalovat a spravovat služby, jako například DNS, Samba, DHCP a další [12].

Webové rozhraní prezentuje zařízení v podobě karet s ovládacími prvky a stavovými informacemi. Nové typy karet lze vytvářet pomocí jazyka HTML, k dispozici je mnoho elementů, od textových polí a tlačítek až po video kontejnery. Na obrázku 5 je vidět ukázkový ovládací panel systému Hass.

2.6 ThingsBoard

ThingsBoard je platforma pro sběr, zpracování a vizualizaci dat a také správu zařízení v prostředí s více zákazníky. Větší část projektu je uvolněna pod open-source licencí, jako komunitní edice, ta umožňuje sbírat data, kategorizovat, ukládat a zobrazovat data, také nabízí integrace s jinými systémy a základní podporu platforem LoRaWAN a SigFox. Sofistikovanější funkce, jako jsou více zákaznické prostředí, pokročilý export data a rozšířenější podporu výše zmíněných platforem, má uživatel k dispozici až v profesionální komerční edici.

Zařízení mohou komunikovat pomocí standardních protokolů MQTT a HTTP, ale na rozdíl od dříve zmíněných komplexních platforem zde neexistuje žádný repozitář šablon zařízení, formát komunikace stanovuje systém ThingsBoard a zařízení jej musí dodržovat. Na rozdíl od jiných platforem je zde věnován důraz na škálovatelnost a odolnost vůči selhání, jednotlivé instance ThingsBoardu mohou operovat jako redundantní cluster.

Důležitou komponentou systému je Gateway, ta je distribuována, jako aplikace v jazyce Java a zpřístupňuje integrace s platformami LoRaWAN a SigFox, umožňuje spojení s externími MQTT brokery a stará se o perzistentní uchování dat. Uživatelská část ThingsBoardu pak disponuje klasickými prvky, jako jsou widgety s velkým výběrem zobrazovacích a ovládacích prvků, stavové panely a také správu a konfiguraci zařízení [15].

2.7 Shrnutí

Výše popsaná řešení patří k těm aktuálně nejzajímavějším a nejpoužívanějším, existuje ale i řada jiných systémů, které se svými vlastnostmi silně podobají představeným řešením, v mnoha případech se jedná o částečně komerční řešení, jako například Samsung SmartThings, Cayenne, AWS IoT, Apple HomeKit, Adafruit IO a další. Existuje také několik menších komunitních projektů jejichž snahou je vzájemné propojení jednotlivých systémů a lepší integrace s hardwarovými moduly, například projekty Open-Home-Automation a Espurna.

Po analýze již existujících řešení jsem stanovil soubor požadavků, které od IoT systému tvořeného primárně moduly ESP8266 a ESP32 očekávám. Specifické požadavky jsou kladeny, jak na software modulu ESP8266 nebo ESP32 (dále jen „zařízení“), tak na řídící aplikaci, která bude data sbírat a vyhodnocovat (dále jen „server“).

Obousměrná komunikace Komunikace mezi serverem a zařízením by měla probíhat primárně ve směru od zařízení k serveru, kdy zařízení bude odesílat pravidelně naměřená data, například stav senzorů, informace o stisknutém tlačítku, nebo přepínači, informace o svém vnitřním stavu. Možná by měla být i komunikace ve směru od serveru k zařízení, kdy server bude mít možnost požádat zařízení o provedení určité akce, například sepnutí spínače, zobrazení určitých dat.

Zobrazení stavu zařízení Uživatel by měl mít k dispozici aktuální stav zařízení. Server by měl zobrazovat zdali je zařízení aktuálně připojeno, popřípadě jak dlouho není připojeno, také by měly být k dispozici základní telemetrické údaje, jako například aktuální IP adresa zařízení, nebo aktuální síla WiFi signálu.

Vizualizace dat Uživatelské rozhraní serveru by mělo poskytnout uživateli přehled přijatých dat v podobě grafů a statistik. Uživatel by měl být schopen upravit formát v jakém data budou zobrazena.

Automatické navázání spojení Automatické přidávání nových zařízení v případě kdy jsou připojena do WiFi sítě. Uživatel by měl mít následně možnost, zdali toto zařízení akceptuje či nikoliv.

Plánované úlohy Server by měl být schopen provádět pravidelné naplánované úlohy. Může se jednat o komunikaci mezi serverem a zařízením, například odeslání příkazů pro daná zařízení. Také by měla existovat možnost vytvořit naplánované úkony pouze na serveru, například vyčištění starých záznamů z databáze.

3 Modul ESP8266 a ESP32

ESP8266 je mikropočítačový modul navržený společností Espressif System v roce 2014, jako jednoduchý převodník sériového rozhraní na WiFi rozhraní. Původně byl modul dodáván s nahraným firmwarem, který poskytoval sadu AT příkazů, které uživateli umožňovali provádět operace, jako připojení k WiFi síti, vytvoření AP, nastavení DHCP, vytvoření TCP serveru, přístup k diagnostickým datům, atd.. ESP8266 bylo obvykle propojeno s nějakým modulem Arduino, nebo podobným zařízením přes rozhraní UART a zprostředkovávalo WiFi konektivitu.

ESP32 je novější revize populárního ESP8266 představená na podzim roku 2016, odstraňuje většinu nedostatků jeho předchůdce a přináší mnoho novinek především množství GPIO vstupů, větší rozmanitost připojitelných periferií, Bluetooth rozhraní a hardware pro akceleraci AES a SSL.

3.1 Specifikace modulů

Vzhledem k tomu, že původním účelem ESP8266 bylo pouze zprostředkovávat bezdrátovou konektivitu, byly u prvních verzí modulu (verze ESP8266-01) vyvedeny pouze 2 GPIO vstupy, pozdější revize měly vyvedeno až 17 GPIO vstupů. Množství vstupů bylo komunitou vnímáno jako jedna z nevýhod, zejména ve srovnání s obdobnými Arduino moduly, které mívají k dispozici i více než dvojnásobné množství vstupů. Mezi další nevýhodu lze zařadit malé množství paměti RAM, 160 kB nepostačuje pro nasazení některých aplikací vytvořených v interpretovaných programovacích jazycích. Nástupce ESP32 má k dispozici větší množství GPIO vstupů a výrazně více paměti RAM, také je vybaven technologií Bluetooth a novými typy senzorů. Přehled rozdílů mezi moduly ESP8266 a ESP32 je k dispozici v tabulce 1 [34][35].

3.1.1 Moduly ESP8266

Většina v současnosti dostupných modulů pochází od výrobce Ai-Thinker, sám Espressif System vytvořil pouze jednu verzi modulu, s názvem ESP-WROOM-02, která ale není příliš rozšířená. Ai-Thinker v průběhu času představil celkem 21 revizí, ESP-01 až ESP-14, které se vzájemně liší zejména počtem vyvedených GPIO vstupů, vnějším provedením, velikostí flash paměti a udělenými certifikacemi [36]. Na obrázku 6 je schématické znázornění jednoho z modulů od výrobce Ai-Thinker, konkrétně revize ESP-12E.

Vzhledem k tomu, že moduly od Ai-Thinkeru mají rozteče výstupů nekompatibilní s nepájivými prototypovacími poli a také z důvodu, že moduly samotné neobsahují obvody potřebné pro napájení, vzniklo mnoho různých vývojových desek. Tyto desky na sobě většinou mají dodatečnou napájecí část, převodník mezi UART a USB rozhraním (pro usnadnění nahrávání firmwaru), popřípadě další periferie, jako například obvody pro připojení baterie, tlačítka, LED diody, senzory a displeje. K nejrozšířenějším patří desky prodávané pod značkou NodeMCU a Wemos s cenou od 2 dolarů. Prodávány jsou i produkty vhodné do produkčního nasazení, například Pro-

Vlastnosti	**ESP8266**	**ESP32**
Procesor (MCU)	Xtensa Single-Core 32-bit L106	Xtensa Dual-Core 32-bit LX6
Frekvence procesoru	80 - 160 MHz	160 - 240 MHz
WiFi konektivita	802.11 b/g/n, HT20	802.11 b/g/n, HT40
Bluetooth	není k dispozici	verze 4.2
SRAM	160 kB	512 kB
Flash paměť (maximum)	16 MB, SPI Flash	16 MB, SPI Flash
GPIO vstupy	17	36
PWM	8 kanálů SW	1 kanál HW, 16 kanálů SW
SPI rozhraní	2x	4x
I2C rozhraní	1x	2x
I2S rozhraní	2x	2x
UART rozhraní	2x	3x
AD převodník	10 kanálů	18 kanálů
DA převodník	není k dispozici	2x 8 b
Ethernet rozhraní	není k dispozici	1x
Sběrnice CAN	není k dispozici	1x verze 2.0
Touch sensors	není k dispozici	10x
Inegrovaný teploměr	není k dispozici	1x
Hallův senzor	není k dispozici	1x
Podpora šifrování	není k dispozici	AES, SHA-2, RSA, ECC, RNG
Napájení	2,5 $\sim$ 3.6 V, $\sim$ 80 mA [2]	2,3 $\sim$ 3,6 V, $\sim$ 50 mA [2]

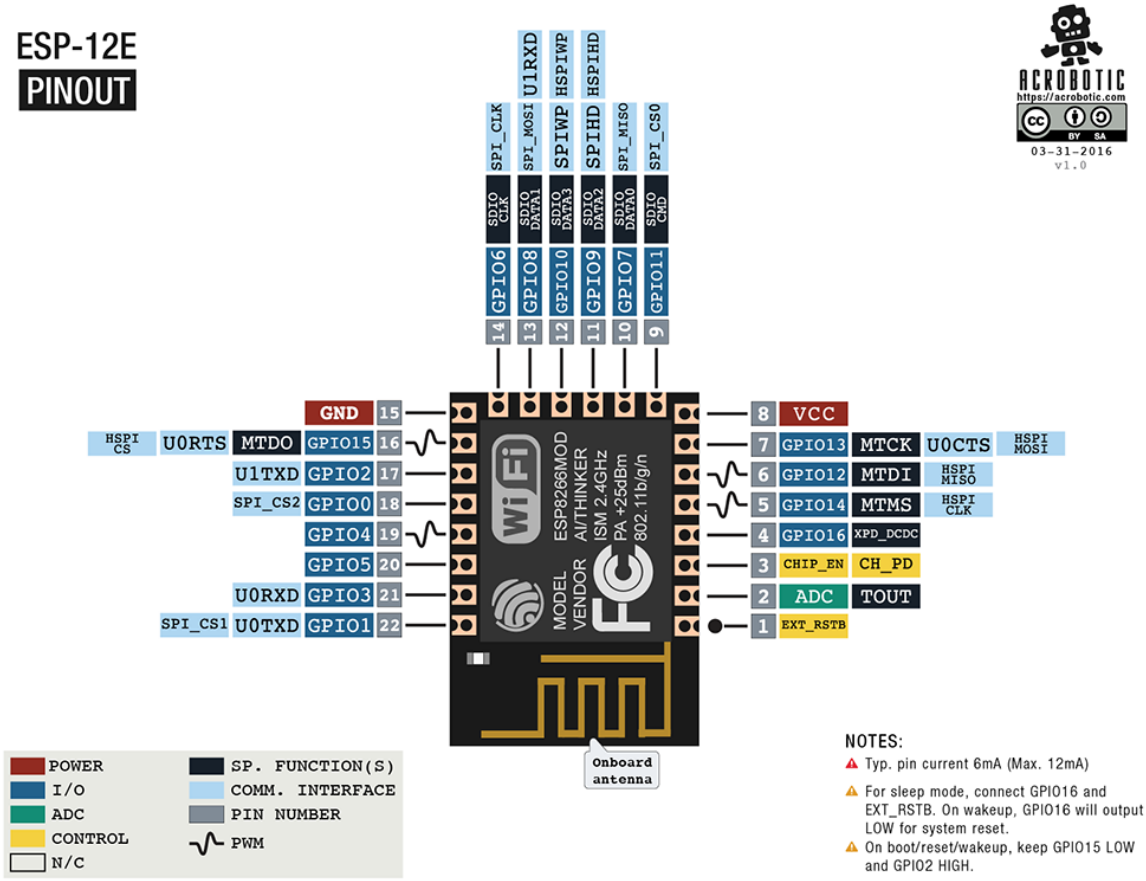
Tabulka 1: Porovnání modulů ESP8266 a ESP32

Dino WiFi-ESP, nebo produkty značky Sonoff, jedná se o programovatelné relé řízené modulem ESP8266.

3.1.2 Moduly ESP32

V případě ESP32 je situace o něco komplikovanější, Espressif System vyrábí několik variant čipu ESP32 a od nich jsou odvozeny stovky různých modulů a vývojových desek od různých výrobců. Samotný Espressif produkuje 14 odlišných vývojových modulů v různých velikostech a s odlišnými přídavnými periferiemi. Čip ESP32 se vyrábí v 5 variantách, které jsou si velmi podobné, jejich přehled je v tabulce 2. Nejzajímavější a nejnovější z variant je ESP32-PICO-D4, jedná se o takzvaný SIP (System in Package), v pouzdře o velikosti 7x7 mm je kromě samotného procesoru umístěn i krystalový oscilátor, obvody pro zpracování signálu a 4 MB plaměti flash [37].

[2] Uvedená spotřeba proudu platí pro průměrné zatížení při nižší frekvenci CPU, při vysoké zátěži a intenzivním používání bezdrátové komunikace může být spotřeba až 500 mA.



Obrázek 6: Schématické znázornění modulu ESP8266 (převzato z [2])

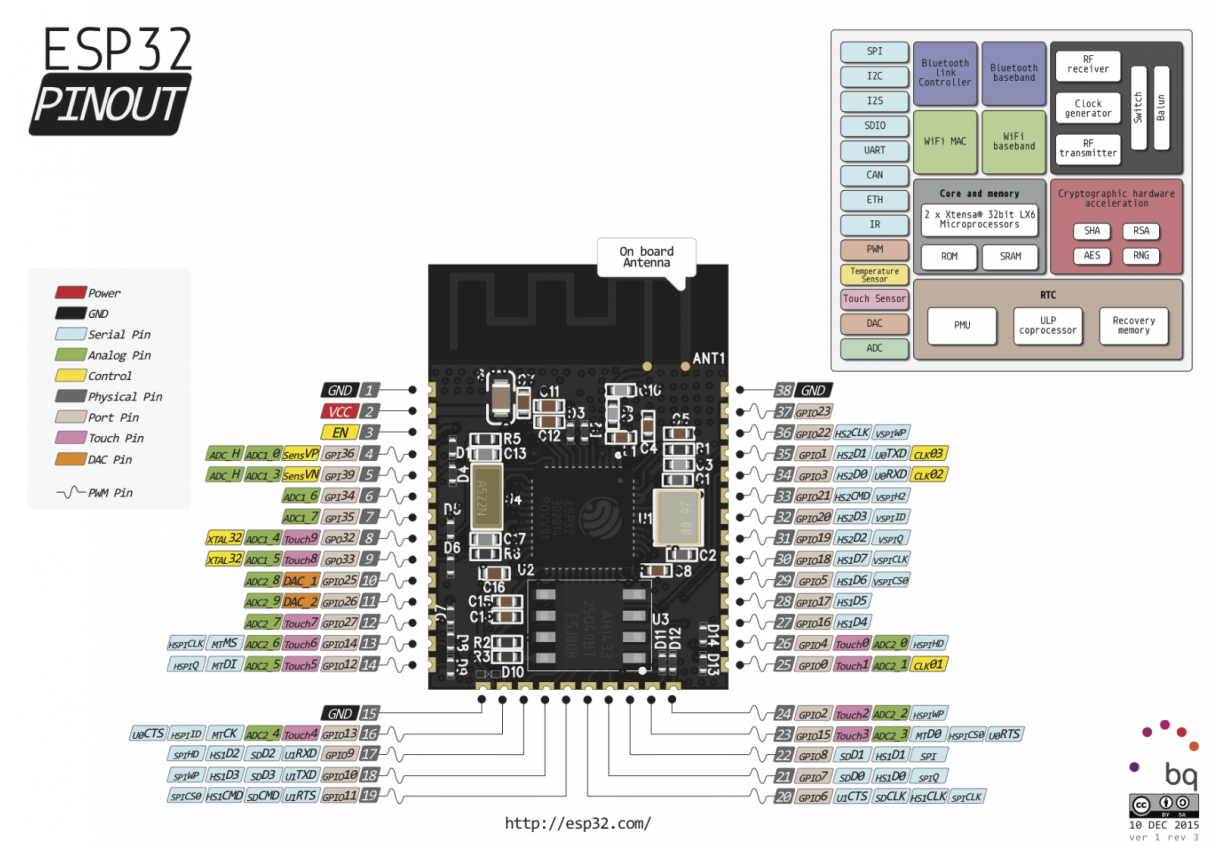
Vlastnosti	D0WDQ6	D0WD	D2WD	S0WD	PICO-D4
Počet jader	2	2	2	1	2
Integrovaná flash	0 MiB	0 MiB	2 MiB	0 MiB	4 MiB
Rozměry	6x6 mm	5x5 mm	5x5 mm	5x5 mm	7x7 mm

Tabulka 2: Porovnání variant čipu ESP32.

Na obrázku 7 je znázornění jednoho z modulů od výrobce Esspressif System, konkrétně ESP-WROOM-32.

3.2 Možnosti vývoje pro ESP8266

Již od počátku si komunita vývojářů uvědomovala, že by se tento relativně výkonný modul dal využít efektivnějším způsobem než pouze jako převodník. Velkou překážkou byla ovšem absence jakéhokoliv vývojového SDK a anglicky psané dokumentace od výrobce. Po rostoucím zájmu o modul ze strany veřejnosti uvolnil výrobce oficiální SDK nástroje pro vývoj aplikací v jazyce C.



Obrázek 7: Schématické znázornění modulu ESP32 (převzato z [3])

Výrobce zveřejnil SDK ve dvou verzích Non-OS SDK a OS SDK založené na operačním systému FreeRTOS. Ani jedno z nich ovšem nepřináší jednoduchý způsob vývoje aplikací, z toho důvodu se komunita vývojářů rozhodla vytvořit vlastní nástroje, které tvorbu softwaru usnadňují a zpřístupňují širší veřejnosti. V současné době je k dispozic celá řada různých SDK pro vývoj softwaru, nejen v C/C++, ale i dalších převážně interpretovaných programovacích jazycích.

3.2.1 Oficiální SDK

Toto SDK je vyvíjeno a udržováno přímo výrobcem Espressif a je poskytováno ve dvou verzích **Non-OS SDK** a **RTOS SDK**, které je založeno na operačním systému FreeRTOS. Non-OS SDK funguje na principu časovačů a volání callbacků, pro vykonávání funkcí v reakci na události. Vzhledem k tomu, že není implementována žádná správa procesů, uživatel musí v programu pravidelně volat obsluhu systémových procesů, které zajišťují spojení s bezdrátovou sítí, obsluhu TCP/IP stacku a dalších.

Verze s FreeRTOS systémem naproti tomu poskytuje prostředí s plánovačem procesů, je možné používat standardní rozhraní tohoto operačního systému, jako je meziprocesní komuni-

kace, synchronizace, uspávání procesů, správa zdrojů a multithreading. Implementováno je také rozhraní soketů. Do SDK je zahrnuta i knihovna cJSON, pro snadné parsování JSON zpráv a mnoho jiných podpůrných knihoven pro přístup k periferiím. Na druhou stranu používání multithreadingu může být na ESP8266 poněkud nebezpečné obzvláště kvůli hardwarovým limitacím modulu a také z toho důvodu, že ne všechny knihovny jsou thread-safe. Vývojářské rozhraní je v mnoha ohledech kompatibilní s rozhraním Non-OS verze, ale na rozdíl od ní RTOS SDK nepodporuje rozhraní AT příkazů [18][19].

Espressif také poskytuje kompletní vývojové nástroje včetně hardwarových modulů a předkompilovaných obrazů částí firmwaru.

3.2.2 Komunitní SDK

Oficiální SDK není vydáno pod open-source licencí a obsahuje několik binárních balíčků bez dostupného zdrojového kódu, z těchto důvodů vyvinula komunita nezávislých vývojářů několik open-source alternativ. Prvním a jedním z nejpoužívanější je **Esp-open-sdk**, v současnosti udržované Paulem Sokolovskym [20]. Toto SDK je založeno na volně šiřitelném toolchainu pro architekturu procesorů Xtensa, na níž je návrh modulu ESP8266 založen, toolchain nazývaný **gcc-xtensa** je vyvíjen Maxem Filippovem [21]. Dále za zmínku stojí SDK **esp8266-devkit** vyvíjené Mikhailem Grigorevem [22].

Je potřeba zmínit, že ani jedna z uvedených variant nepřináší příliš pohodlný způsob pro vývoj programů, pro kompilaci programu je nutno mnoho nástrojů a zprovoznění vývojového prostředí tedy trvá i několik hodin. Vývoj aplikací rovněž vyžaduje přinejmenším pokročilé znalosti programování v jazyce C a znalosti architektury mikropočítačů.

3.2.3 Arduino Core

Pravděpodobně nejpopulárnějším možností při vývoji aplikací pro ESP8266 je použití **Arduino Core** SDK. Tento projekt je udržován komunitou a integruje vývojové nástroje pro moduly ESP8266 do vývojového prostředí Arduino IDE. Především ale umožňuje programovat moduly ESP8266 podobným způsobem, jako mikropočítače platformy Arduino, je tak možné používat většinu existujících knihoven pro Arduino. Interoperabilita mezi platformami byla v počátcích vývoje pro vývojáře velkou výzvou, architektura Xtensa je totiž v mnoha ohledech nekompatibilní s architekturou AVR užívanou většinou mikropočítačů Arduino. Díky velkému úsilí komunity a intenzivnímu vývoji je dnes většina knihoven pro Arduino kompatibilní i s moduly ESP8266 [24].

Arduino přináší velmi pohodlné vývojové rozhraní vhodné i pro uživatele s malými zkušenostmi s vývojem softwaru pro mikropočítače. Podstatnou výhodou je také přístup ke stovkám knihoven pro ovládání různých periferií a komunikaci pomocí široké škály protokolů. Velká část ESP modulů je navíc dostupná skrze manažér desek v prostředí Arduino IDE a podporována je také platforma PlatformIO [23].

3.2.4 NodeMCU Lua firmware

NodeMCU je implementace dynamicky typovaného programovacího jazyka Lua pro mikropočítače ESP. Firmware byl původně vyvíjen společně se specializovanými hardwarovými moduly NodeMCU, což byly klasické ESP8266-12E moduly s přidanými periferiemi pro snadnější nahrávání firmwaru. Vývoj a prodej hardwarovým modulů byl nicméně již ukončen, ale vzhledem k tomu že návrh desky byl uvolněn pod svobodnou licencí, je stále k dostání mnoho kompatibilních hardwarových modulů.

Lua firmware je udržován komunitou, která spravuje celou řadu knihoven pro různé periferie a protokoly. Uživatel si obraz firmwaru sestaví dle svých potřeb skrze webové rozhraní, ve kterém si může vybrat, jaké konkrétní knihovny si přeje využívat [25].

Programy v Lue jsou vytvářeny metodou reakcí na události, které generují různé periferie, nebo vnitřní časovače. Vývoj programů také usnadňuje integrace se souborovým systémem SPIFFS, díky němu je možné skripty v Lue jednoduše zapisovat do flash paměti a také k ní programově přistupovat.

Bohužel i po letech vývoje existují problémy se stabilitou a spolehlivostí firmwaru a to zejména kvůli nedostatku dostupné paměti na ESP8266, z těchto důvodu je Lua firmware určen spíše pro experimentální účely [26].

3.2.5 Mongoose OS

Tento firmware je postaven nad oficiálním firmwarem Espressifu, ale velmi výrazně jej rozšiřuje. Velký důraz je zde kladen na integraci s platformou AWS IoT a podobnými komerčními platformami. Tvůrci se snaží systém cílit na komerční použití, kromě silné integrace s platformou AWS IoT je k dispozici i řada funkcí spojených s bezpečností, vzdálenou správou zařízení a OTA aktualizacemi[3]. Cesanta, společnost stojící za vývojem Mongoose OS, nabízí rovněž komerčně licencovanou verzi tohoto firmwaru [27].

Uživatel systému k vytváření programů může používat jazyky C, C++ nebo JavaScript. Pro snadnější vývoj je připraven plugin do vývojového prostředí Visual Studio Code. Programy v JavaScriptu je možné vytvářet a nahrávat do zařízení z vývojového prostředí ve webovém prohlížeči.

3.2.6 Ostatní

K dispozici je ještě několik menších projektů, které umožňují vyvíjet programy v různých dynamicky typovaných programovacích jazycích.

[3]Over-the-air programming je metoda doručování softwaru po bezdrátovém přenosovém kanálu. Důležitým aspektem OTA aktualizací je centrální řízení, pro úspěšné doručení aktualizace tak není nutný žádný zásah uživatele na straně zařízení.

MicroPython Odlehčená implementace programovacího jazyka Python pro mikropočítačové zařízení. Nadace vyvíjející MicroPython vytváří i vlastní moduly nazvané Pyboards, ale podporované jsou i zařízení ESP8266 a ESP32. Stabilita systému je v současné době na dobré úrovni nicméně dostupnost knihoven pro periferie je prozatím omezená [28].

Espruino Vývojová platforma založená na JavaScriptu. Opět jsou k dispozici i samostatné vývojové moduly. Aplikace uživatel vyvíjí za pomocí rozšíření do webového prohlížeče Chrome, z něho jsou programy také nahrávány na zařízení, podobně jako na flash paměťové úložiště [29].

Sming Framework pro vývoj aplikací v C++ založený na Non-OS oficiálním SDK. Integruje mnoho užitečných knihoven a výrazně tak usnadňuje uživateli vývoj aplikací [30].

ESP Basic Interpret jazyka Basic pro mikropočítač ESP8266. Programování je možné v editoru přímo na zařízení, skrze vzdálený přístup přes prohlížeč [31].

3.2.7 Shrnutí

Po analýze a otestování většiny dostupných řešení jsem došel k závěru, že pravděpodobně nejvhodnějším nástrojem k vývoji aplikací pro ESP8266 je Arduino Core SDK. Arduino Core totiž nabízí zdaleka nejširší podporu různých hardwarových modulů a knihoven třetích stran, zároveň umožňuje tvorbu stabilních aplikací v pohodlném vývojářském prostředí.

Na druhou stranu není Arduino Core vhodné, pokud má uživatel v úmyslu manipulovat s hardwarem na „nízké úrovni“, například provozovat zařízení s ohledem na minimalizaci spotřeby energie. Pro tyto účely se jeví, jako vhodnější použití rozsáhlého komunitního SDK Esp-open-sdk.

Pro rychlý vývoj prototypů lze s výhodou využít i prostředí NodeMCU, přes nižší stabilitu a množství rozšiřitelných knihoven, nabízí stále velmi pohodlné prostředí pro tvorbu jednodušších aplikací.

3.3 Možnosti vývoje pro ESP32

V případě ESP32 zvolil výrobce trochu odlišný přístup, po zkušenostech s předchozím modelem výrazně zvýšil úsilí při tvorbě dokumentace a SDK. Espressif v současnosti vyvíjí a udržuje 2 verze SDK jako open-source. První verze SDK je nazývána Espressif IoT Development Framework [16], která kromě samotného SDK napsaného v C a C++ zahrnuje i nástroje pro kompilaci a nahrání kódu do modulu. Druhá verze nazvaná Arduino Core for ESP32 WiFi chip [17], je určena pro platformu Arduino a poskytuje pro uživatele příjemnější metodu programování modulů. Na rozdíl od ESP8266 jsou obě tato SDK udržovaná Espressifem, obě verze jsou open-source a obě jsou založeny na operačním systému FreeRTOS, původní Non-SDK varianta z ESP8266 již není k dispozici.

Kromě oficiálních SDK je k dispozici i řada nástrojů pro ESP8266, které po příchodu ESP32 na trh přidaly podporu pro tento modul. Jedná se o řešení: NodeMCU, Mongoose OS, MicroPython, Espruino. Pro ESP32 vzniklo dokonce více implementací programovacího jazyka Lua, jedná se o **LuaNode** [32] a **Lua-RTOS-ESP32** [33].

3.3.1 Shrnutí

Podobně jako u ESP8266 i zde se kloním k názoru, že Arduino Core je nejlepší variantou pro vývoj programů pro tento mikropočítač, přináší totiž výhodu velkého množství knihoven dostupných pro platformu Arduino. Vzhledem k tomu, že finální verze ESP32 byla uvolněna teprve nedávno, nejsou zde prozatím implementovány všechny funkce dostupné u předešlého modelu, vývoj v této oblasti je však velmi intenzivní.

4 Protokol MQTT

Protokol MQTT je komunikační protokol navržený pro zajištění spolehlivé komunikace mezi zařízeními s minimálními požadavky na výkon zařízení a kvalitu přenosového kanálu. MQTT protokol vznikl v roce 1999 ve společnosti IBM, jeho původním účelem bylo zprostředkovat komunikaci senzorům monitorujících ropovody. Tyto senzory byly spojeny s okolním světem přes síť satelitů, která je obecně velmi pomalá a nepříliš spolehlivá. Monitorovací senzory byly napájeny bateriemi, bylo tedy nutné zajistit minimální nároky na výkon a tedy spotřebu baterie. Když byl v roce 2010 protokol MQTT uvolněn pod royalty-free licencí stal se díky svým vlastnostem populární v oblasti IoT, jeho vlastnosti se shodují s požadavky na výstavbu IoT systémů [43].

MQTT pracuje nad protokolem TCP a broker standardně naslouchá na portu 1883, pro zabezpečené spojení TLS pak na 8883, existuje i implementace pracující přes technologii WebSockets, obvykle provozovaná na portech 8080, nebo 8081, porty lze však libovolně měnit.

MQTT je postaveno na konceptu publisherů (publikujících) a subscriberů (odebírajících), publisher odešle zprávu s určitým tématem a všichni subscribeři, kteří dané téma odebírají zprávu obdrží. Zařízení může odebírat libovolné množství témat a zároveň do nich publikovat zprávy, odběr zpráv ale není podmínkou pro možnost publikování zpráv.

Obsah zpráv není standardem nijak pevně definován, data jsou přenášena binárně a výchozí maximální velikost jedné zprávy je 256 MB, toto maximum lze ovšem překročit pokud s tím počítá broker [39].

Centrálním prvkem protokolu je MQTT broker, jeho účelem je přijímat zprávy od publisherů a odesílat je subscriberům, popřípadě je ukládat do fronty. Předtím než klient začne publikovat nebo odebírat zprávy, musí se u brokeru zaregistrovat unikátní klientským ID[4], popřípadě může použít uživatelské jméno a heslo, pokud to broker vyžaduje. Dle nastavené QoS úrovně taky dohlíží na to, že zprávy byly skutečně přijaty a korektně odeslány. Existuje několik rozšířených implementací brokeru, pravděpodobně nejpopulárnější je verze Eclipse Paho vyvíjená nadací Eclipse Foundation, pod BSD licencí.

4.1 Téma zpráv

Téma (topic) zprávy je hierarchická stromová struktura, která slouží ke kategorizaci zpráv, jednotlivé úrovně tématu jsou od sebe odděleny lomítkem `/`. Téma není nutné nijak konfigurovat nebo vytvářet, ve chvíli kdy je brokerem přijata zpráva, nebo odběr z nového neznámého tématu, automaticky téma vytvoří.

Standard nedefinuje žádnou pevně danou strukturu témat, záleží na uživateli, nebo na konkrétní aplikaci jak témata definuje. Strom témat lze navrhnout třeba na základě významu dat která jsou v daném tématu přenášena, nebo na základě geografické polohy zařízení, které zprávy

[4]Ve většině případů nemusí klient unikátní ID definovat, je mu totiž vygenerováno automaticky. V případě že si ho vytvoří sám, musí zajistit aby se pod stejným ID nepokusil připojit vícekrát.

odesílá. Správně definovaná struktura téma může zefektivnit odběr témat, subscribeři nebudou muset odebírat témata s daty o které nemají zájem. Příkladem tématu může být například: `iot/sensors/weather-station/temperature`, nebo `home/basement/sensor1/humidity`.

Při odběru tématu může subscriber použít zástupné znaky, takzvané wildcard symboly `+` a `#`. Znak `+` nahrazuje právě jednu úroveň v hierarchii tématu, použijme již zmíněný příklad `iot/sensors/+/temperature`, znak `+` zde nahrazuje úroveň `weather-station`, místo ní zde může být jakýkoliv jiný řetězec, které neobsahuje `/`, tedy i prázdný řetězec. Wildcard symbol `#` nahrazuje jednu a více úrovní ve struktuře tématu, například `iot/sensors/#` představuje všechny témata začínající na `iot/sensors/` a následovány libovolným dalším zanořeným tématem, do odběru jsou zahrnuty i zprávy v samotném tématu `iot/sensors`. Pokud použijeme wildcard `#` musí se vždy jednat o poslední znak v registrovaném tématu, téma `iot/sensors/#/temperature` je tedy nevalidní a nelze jej zaregistrovat.

Zvláštní funkci má také znak `$`, pokud téma začíná tímto znakem jedná se o téma se speciálním významem, příkladem mohou být témata `$SYS/`, které publikuje sám broker a mají pevně danou strukturu [38].

4.2 QoS zpráv

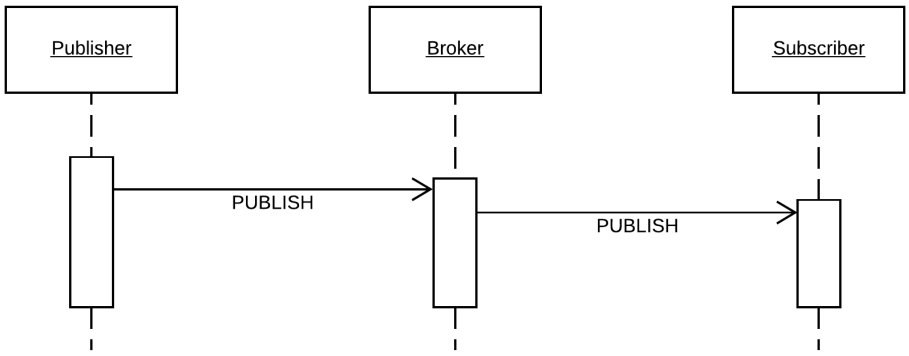
Na spolehlivost doručení zpráv byl při návrhu protokolu kladen velký důraz, proto byly v rámci protokolu definovány 3 úrovně garance doručení zprávy.

Ve chvíli kdy broker přijme zprávu pokouší se ji všem odběratelům doručit se stejnou QoS úrovní, s jakou zprávu přijal, nicméně subscriber může při odběru tématu indikovat s jakou QoS úrovní si přeje, aby mu zprávy byly doručeny. Pokud publisher odešle zprávu s vyšší úrovní QoS, než odběratel očekává je zpráva před doručením odběrateli degradována na požadovaný QoS [44].

Pokud je zpráva odeslána za úrovni 1, nebo 2 broker zprávy shromažďuje i pro odběratele, kteří jsou momentálně odpojeni, podmínkou je, že musí mít nastavený příznak persistent session.

QoS 0 Označovaný také jako „nejvýše jednou“ (at most once), negarantuje doručení zprávy brokeru. Zpráva je publisherem odeslána, ale nečeká se na žádnou odpověď od brokeru, může se tedy stát že se zpráva při přenosu ztratí a odesilatel se o tom nedozví. Na obrázku 8 je zachycen proces výměny zpráv.

Tato úroveň se používá, pokud máme data, která chceme doručit přístupem best-effort, například obrazová data, u kterých příliš nevadí pokud se několik snímků nebo řádků ztratí při přenosu, naopak je žádoucí aby zpracování bylo pokud možno nejrychlejší.

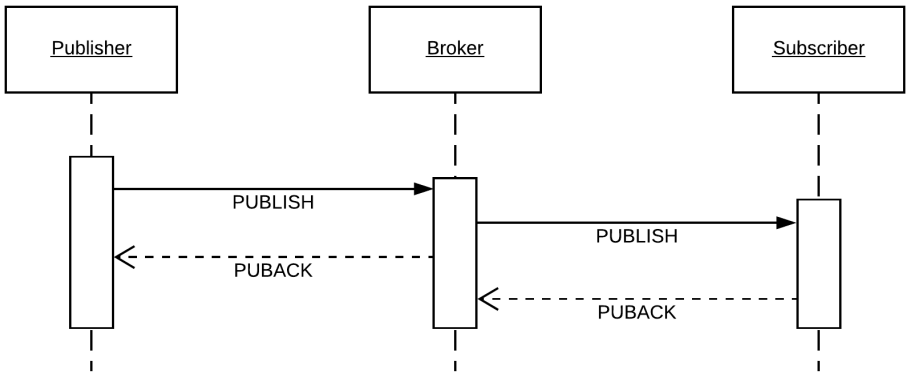


Obrázek 8: Diagram výměny zpráv v protokolu MQTT na úrovni QoS 0.

QoS 1 Garantuje doručení alespoň jednou (at least once). Publisher odešle zprávu `PUBLISH` brokeru, ve chvíli kdy broker zprávu obdrží odešle publisherovi potvrzení `PUBACK`, publisher na toto potvrzení čeká a smaže zprávu ze své fronty neodeslaných zpráv až ve chvíli kdy skutečně obdrží potvrzení `PUBACK`. Pokud potvrzení neobdrží, odešle po chvilce (závisí na nastavení odesilatele) zprávu znovu se značkou DUP (duplicate flag), která oznamuje, že se zprávu již jednou nepodařilo doručit. Kvůli tomuto chování může dojít k vícenásobnému odeslání zprávy, potvrzení `PUBACK` se může po cestě ztratit a publisher se tedy bude domnívat, že první zpráva na broker nedorazila. Sekvence výměny zpráv je zachycena na obrázku 9.

Stejné chování je použito i ve chvíli kdy broker distribuuje zprávu svým subscriberům. Broker odešle zprávy odběratelům a očekává potvrzení `PUBACK`, až ve chvíli kdy potvrzení obdrží od všech odběratelů smaže zprávu z fronty zpráv.

QoS 1 se používá jako optimální kompromis mezi ostatními úrovněmi, vhodným použitím může být odesílání naměřených dat, nebo stavu. S QoS 1 máme garanci že data budou doručena minimálně jednou, pokud by chybou komunikace byla data doručena vícekrát, nezpůsobí to závažné problémy.

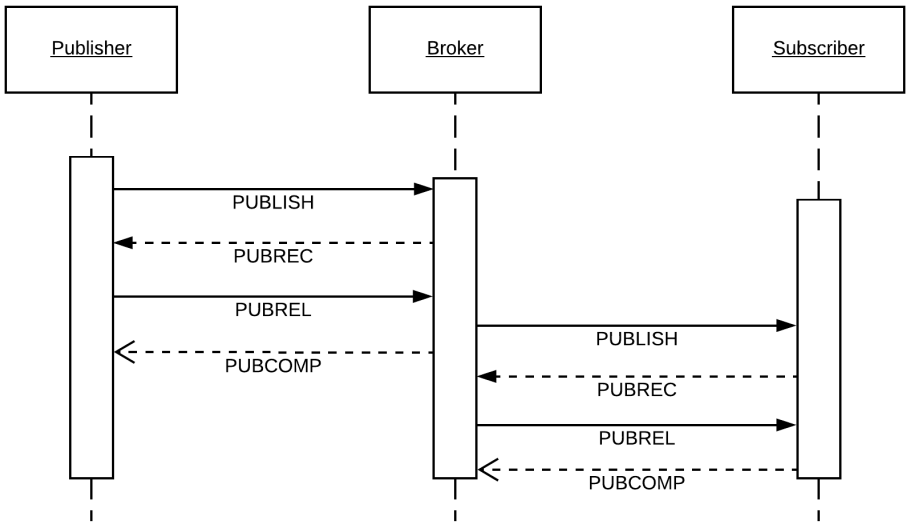


Obrázek 9: Diagram výměny zpráv v protokolu MQTT na úrovni QoS 1.

QoS 2 Zajistí doručení zprávy právě jednou (exactly once), nemůže tedy dojít k duplicitnímu doručení jedné zprávy. Na obrázku 10 můžeme vidět sekvenci zpráv, na počátku publisher odešle zprávu `PUBLISH` a očekává od brokeru potvrzení o přijetí `PUBREC`, pokud potvrzení nedorazí opakuje odeslání `PUBLISH`. Po přijetí potvrzení `PUBREC` odešle publisher povolení k odeslání `PUBREL`, dokud broker povolení `PUBREL` neobdrží uchovává přijatou zprávu ve frontě zpráv čekajících k odeslání. Po příjetí zprávy `PUBREL` broker odešle zprávu všem odběratelům a odešle potvrzení o dokončení `PUBCOMP` zpět publisherovi, ten si po přijetí potvrzení smaže zprávu ze své fronty. V případě, že `PUBREL` nedorazí včas na broker, odešle broker opakovaně potvrzení `PUBREC`, stejná situace platí i pokud publisher neobdrží zprávu `PUBCOMP`, po čase znovu zašle zprávu `PUBREL`.

Scénář je obdobný i u rozesílání zpráv odběratelům, broker zde hraje roli odesilatele prvotní zprávy `PUBLISH`.

Tato úroveň spolehlivosti nachází využití například u zpráv, které indikují příjemci změnu stavu, obvykle je nežádoucí vynutit si změnu stavu omylem vícekrát. QoS 2 může být vhodný i pro zasílání kritických dat, příkladem může být odesílání aktuálních otáček, nebo stav zavírání a otevírání ventilu, špatná detekce těchto stavů by mohla způsobit u kritických systémů závažnou havárii. Nevýhodou je pochopitelně režie způsobená vícenásobným potvrzováním zpráv, musíme tedy vždy najít vhodný kompromis mezi spolehlivostí a rychlostí doručení.



Obrázek 10: Diagram výměny zpráv v protokolu MQTT na úrovni QoS 2.

4.3 Další funkce

Protokol MQTT disponuje několika doplňkovými funkcemi, které mohou najít využití ve zvláštních případech.

Retain zpráva Publisher může při odesílání zprávy do určitého tématu nastavit tento retain příznak, který brokeru indikuje, že nemá zprávu mazat ze své fronty po jejím odeslání. Subscriber, který se přihlásí k odběru tématu obdrží okamžitě po přihlášení poslední uloženou retain zprávu. Automatické odeslání retain zpráv funguje i v kombinaci s wildcard značkami, pokud je retain zpráva uložena v tématu `iot/sensors/weather-station/#` obdrží ji odběratel i při registraci tématu `iot/sensors/weather-station/temperature`. Pro každé téma udržuje broker vždy jen jednu poslední přijatou retain zprávu.

Retain zprávy se dají využít například pro uložení poslední naměřené hodnoty, případný odběratel obdrží hodnotu ihned po registraci odběru a nemusí čekat než zařízení vygeneruje novou hodnotu.

Last will V češtině označována jako závěť, tuto zprávu broker rozešle v případě, že dojde k nedobrovolnému odpojení klienta, například z důvodu ztráty konektivity, pokud se klient odpojí dobrovolně pomocí volání `DISCONNECT` zpráva last will se nerozesílá. Klient si musí tuto funkcionalitu aktivovat tím, že definuje obsah zprávy po připojení k brokeru. Při aktivaci musí klient specifikovat téma do kterého se bude last will zpráva rozesílat, obsah a QoS, může být použit také retain příznak.

Příkladem užití je například rozeslání varovné notifikace v případě, že se zařízení odpojí náhle od sítě a nestačí se včas korektně odpojit.

MQTT-SN Nebo také MQTT for Sensor Networks, jedná se o rozšíření protokolu MQTT, které umožňuje komunikaci pomocí protokolu UDP, snaží se tak protokol ještě více odlehčit a zpřístupnit jej pro opravdu velmi málo výkonná zařízení, popřípadě pro real-time aplikace kde i užití protokolu TCP vytváří příliš velké zpoždění [40]. Tento protokol je relativně nový (první návrh v roce 2013) a jeho podpora v současných implementací brokerů a klientských knihoven je docela slabá a prozatím se nedočkal výrazné popularity.

MQTT-SN zavádí novou úroveň QoS -1, na této úrovni je zpráva odeslána brokeru, bez nutnosti navázání spojení s brokerem. Zpráva je jednoduše odeslána pomocí UDP na adresu a port brokeru, odesilatel nekontroluje, zdali broker existuje a naslouchá na adrese, ani jestli dané téma na serveru existuje. Rozšíření přináší také zjednodušené témata, která jsou místo textového řetězce reprezentována pouze číslem, toto téma však musí být předkonfigurováno na brokeru.

Zajímavou funkcionalitou je broker discovery, broker pomocí multicastu rozesílá informace o své existenci, klienti mohou specifické multicastové adrese naslouchat a vyhledat tak dostupné brokery. Standard ovšem nedefinuje žádnou konkrétní multicastovou adresu a port pro rozesílání discovery zpráv, což využití této funkcionality lehce komplikuje.

4.4 Autentizace a zabezpečení

MQTT protokol poskytuje několik možností jak autentizovat připojené klienty. První metodou je použití uživatelského jména a hesla. Broker spravuje seznam uživatelů a pokud chce klient s brokerem komunikovat musí při navazování předat své jméno a heslo. V nastavení brokeru může být definováno, k jakým tématům bude mít klient přístup. Heslo je mezi klintem a brokerem přenášeno v nezašifrované podobě, lze jej tedy odposlechnout.

Uživatele lze také autentizovat pomocí klientského ID, které je povinné. Broker může mít nastaveno ACL pro určitá klientská ID, je tak možné zpřístupnit téma pouze klientům jejichž ID začíná například specifickým prefixem.

Poslední metodou autentizace je použití certifikátu, tato metoda přináší nejvyšší zabezpečení zároveň je však nejobtížnější na implementaci. Pro každého klienta brokeru je potřeba vygenerovat nový certifikát a zařízení ho předat. U velkých dynamických systémů je tento postup těžko realizovatelný.

Autentizace sama o sobě nezajišťuje žádné zabezpečení přenášených dat, ty jsou ve výchozím nastavení přenášeny ve zcela nešifrované podobě. Zabezpečení je možné provést na dvou úrovních, šifrování obsahu zprávy a šifrování celého spojení. Pro šifrování obsahu zpráv lze použít mnoho standardních metod symetrického a asymetrického šifrování. Tato varianta dokáže zabezpečit obsah zpráv, ale nedokáže zabezpečit řídící zprávy, tedy například přihlašovací heslo a jméno.

Pro zabezpečení celého spojení se používá protokol TLS, který zajistí šifrované spojení na úrovni protokolu TCP/IP. Takovýmto kanálem je možné bezpečně přenášet i přihlašovací údaje. Bohužel ne všechny implementace MQTT klientů podporují použití TLS zabezpečení, zvláště ty určené pro méně výkonná zařízení [45].

5 EspHub

Na základně vlastností již existujících řešení a analýzy požadavků na systém pro domácí automatizaci, jsem se v rámci této práce pokusil navrhnout systém, který bude do jisté míry pokrývat nedostatky existujících řešení. Vzhledem k rozsahu práce není v mých silách pokrýt všechny možné případy užití, přesto jsem se pokusil navrhnout systém, který bude co nejuniverzálnější a případně rozšiřitelný. Systém jsem nazval **EspHub** a budu na něj v následujícím textu tímto názvem odkazovat.

EspHub se skládá z několika komponent, které plní zcela odlišné role. První a nejpodstatnější komponentou je aplikace **EspHubServer**, tato komponenta plní roli serveru, komunikuje s MQTT brokerem a sbírá data a telemetrie odeslané ze zařízení, nasbíraná data ukládá do databáze, server se stará také o objevování nově připojených zařízení, spravuje plánovač úloh a zprostředkovává uživatelské rozhraní.

EspHubLib je knihovna pro moduly ESP8266 a ESP32, která usnadňuje komunikaci se serverem a umožňuje modulům automatické připojení k lokálnímu EspHubServeru.

EspHubUnilib je sada nástrojů vytvořených v programovacím jazyce Python. Nástroj Unilib dokáže částečně emulovat chování knihovny EspHubLib na platformě Linux, mezi další nástroje patří například ImageTransmitter, který dokáže odesílat obrazová data na displeje připojené k modulům ESP8266, nebo ESP32.

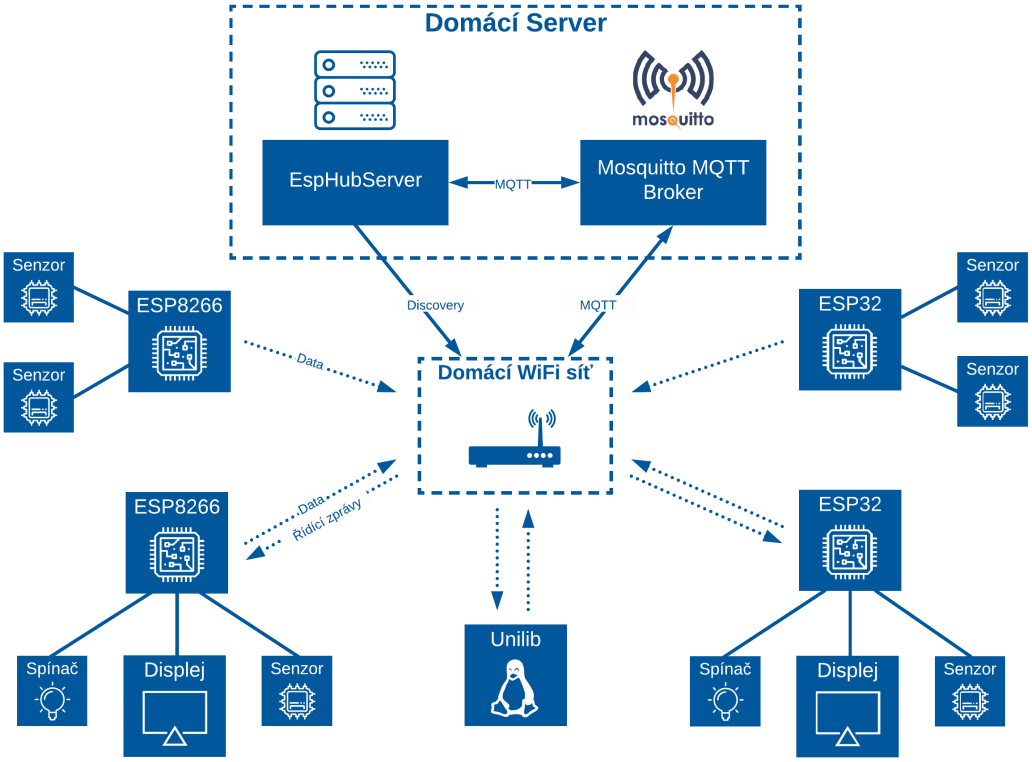
Obrázek 11 znázorňuje princip fungování systému EspHub. Na vrcholu hierarchie jsou EspHubServer a MQTT broker, kteří řídí komunikaci přes komunikační kanál, kterým je WiFi síť. K WiFi síti jsou připojeny také ESP moduly a popřípadě instance EspHubUnilib, kteřé komunikují pomocí MQTT zpráv se serverem.

5.1 Funkce systému

Na základě analýzy existujících řešení jsem stanovil 7 základních vlastností, kterými by měl systém pro domácí automatizaci disponovat. V této kapitoly budou tyto vlastnosti stručně popsány.

5.1.1 Autodiscovery

Hlavním nedostatkem všech zkoumaných řešení byla absence jakéhokoliv automatizovaného mechanizmu, jak připojit nově naprogramované zařízení ESP8266, nebo ESP32 k domácímu automatizačnímu serveru. Tento nedostatek jsem se v mnou navrženém systému pokusil odstranit a vytvořil jsem proces nazvaný autodiscovery protokol, ten detekuje nově připojené zařízení v lokální WiFi síti, poskytne mu informace o dostupném serveru a vyzve uživatele, aby přidání nového zařízení potvrdil.



Obrázek 11: Přehled funkce systému EspHub

5.1.2 Archivace dat

Při registraci zařízení k serveru, zařízení oznamuje jaké data bude serveru poskytovat, tato data jsou označována, jako vstupní ability. Abilita může být například připojený senzor, stav GPIO, nebo teplota čipu, závisí čistě na uživateli, jaké periferie k zařízení připojí a jaká data bude posílat. Standardně by měla být data odesílána ve formátu jednoho čísla, ale přípustné jsou i hodnoty, jako řetězec znaků, nebo data ve formátu JSON. Odeslaná data jsou následně odchycena serverem, který je archivuje do databáze. Veškerá komunikace mezi zařízením a serverem probíhá přes protokol MQTT a může být zabezpečena standardními metodami dostupnými v protokolu MQTT.

5.1.3 Vizualizace dat

Server zprostředkovává uživateli přehled nad archivovanými daty pomocí webového rozhraní, které je schopno vizualizovat uložená data do podoby grafů. Rozhraní je také schopno v reálném čase zobrazovat aktuální stav sledovaných hodnot.

5.1.4 Telemetrie

Aby měl uživatel přehled nad stavem svých zařízení, odesílají zařízení v pravidelných intervalech telemetrická data o svém stavu. Zařízení ESP8266 a ESP32 odesílají svoji aktuální IP adresu, SSID přístupového bodu WiFi, ke kterému jsou připojeny, RSSI, MAC adresu, stav paměti a hladinu elektrického napětí na procesoru. Pokud se nějaké ze zařízení v definovaném intervalu neodešle zprávu s telemetrií, takzvaný keepalive, notifikuje server uživatele, že je zařízení nedostupné.

5.1.5 Ovládání zařízení

Zařízení mohou data nejen odesílat, ale mohou je i přijímat, při registraci zařízení k serveru může zařízení oznámit, že je schopno zpracovávat příchozí data, takzvané výstupní ability[5]. Uživatel si na zařízení definuje funkci, která bude reagovat na zaslání informací o konkrétní výstupní abilitě a v návaznosti na přijetí zprávy může vykonat na zařízení akci. V tuto chvíli jsou v systému předdefinovány 3 různé typy výstupních abilit: tlačítko, přepínač a displej. Pro tyto ability je v uživatelském rozhraní serveru připraveno příslušné ovládání.

5.1.6 Zobrazování na zařízení

Zvláštní kategorií výstupních ability jsou pak displeje, pokud zařízení disponuje displejem může uživatel definovat libovolný obsah, který se následně vykreslí na připojeném displeji. Obrazová data jsou primárně přenášena přes protokol MQTT, ale volitelně je připravena i možnost přenosu přes protokol UDP. V aktuální verzi systému jsou podporovány pouze modely displejů SSD1306 a částečně ILI9341. Další typy mohou být přidaný pokud uživatel definuje metody, pomocí kterých bude obraz na straně serverů připravován.

5.1.7 Plánování úloh

Uživatel může naplánovat pravidelné akce, které se budou v určitých intervalech provádět na serveru, nebo zařízení. K tomuto účelu slouží plánovač úloh na serveru, který na základě záznamu v databázi vytvoří při spuštění aplikace frontu naplánovaných úloh, ty jsou následně, v zadaný čas, paralelně vykonávány. Současná verze systému umožňuje přes uživatelské rozhraní definovat pouze plánované zobrazování obsahu na displeji zařízení, ale jiné typy plánovaných úloh mohou být snadno implementovány.

5.2 Komunikace

Jednou z nejdůležitějších otázek při návrhu systému bylo jakým způsobem budou zařízení a server navzájem komunikovat. Protokol zajišťující komunikaci musí být spolehlivý, robustní a

[5]Terminologie vstupních a výstupních abilit je z pohledu serveru. Vstupní ability (input ability) jsou data, která přichází serveru ze zařízení. Výstupní ability (output ability) jsou data zasílaná serverem na zařízení

zároveň nenáročný na implementaci, zařízení ESP mají limitovanou výpočetní kapacitu a paměť, proto by použitý protokol neměl klást příliš velké nároky na výkon.

V počátcích vývoje byl zvažován například protokol REST, ten se ovšem neukázal jako nejvhodnější a to zejména kvůli komplikované implementaci zpětného komunikačního kanálu (ze serveru na zařízení). Zařízení ESP může v jeden okamžik zpracovávat pouze jednu úlohu, nutnost pravidelně naslouchat na příchozí REST zprávy by vedla k větším nárokům na výpočetní čas a spotřebu energie. Z celkového pohledu je REST zbytečně komplikovaný a „těžký“ protokol pro účel výměny krátkých a jednoduchých zpráv.

Další zvažovanou alternativou byla vlastní implementace komunikačního protokolu nad protokolem TCP, popřípadě UDP. První verze systému dokonce tento přístup využívaly, server naslouchal na otevřeném TCP portu, zařízení navázalo na tento port spojení a odeslalo data, v otevřeném TCP spojení se následně mohli přenášet i data v opačném směru, od serveru k zařízení. Toto řešení bylo funkční, ale především implementace serverové části byla poměrně komplikovaná a nedosahovala spolehlivosti jiných již existujících řešení.

Pro komunikaci byl nakonec použit protokol MQTT, ten je již od počátku navržen tak, aby byl velmi spolehlivý a škálovatelný a také s ohledem na nízký výkon zařízení, je tedy velice vhodný pro použití v oblasti internetu věcí a vyhovuje všem nárokům, které jsem stanovil pro systém EspHub. MQTT protokol disponuje centrálním prvkem brokerem, který zpracovává všechny zprávy. V systému EspHub každé zařízení publikuje zprávy pod svým unikátním tématem, témat do kterých zařízení publikuje může být i více. Server následně odebírá zprávy ze všech zařízení, které uživatel v systému registroval. Server může také odeslat zařízení zprávu, všechna ESP zařízení s knihovnou EspHubLib odebírají téma, které slouží jako kanál pro řídící zprávy, uživatel může poté definovat reakce na příchozí zprávu.

V protokolu MQTT lze nastavit několik úrovní QoS, nejnižší 0 negarantuje žádné doručení zprávy a nejvyšší 2 zaručuje že zpráva dorazí na broker vždy právě jednou. Většina zpráv v systému EspHub je odesílána na QoS 1, což znamená že zpráva dorazí na broker minimálně jednou, tento stupeň kvality byl zvolen s ohledem na časovou náročnost zpracování zpráv na úrovni 2. Jedinou výjimkou jsou zprávy obsahující obrazová data, u kterých se používá úroveň 0, u těchto zpráv je důležité, aby na zařízení dorazili pokud možno nejrychleji zkutečnost, že některé zprávy mohou být zahozeny nepředstavuje problém.

5.2.1 Formát zpráv

Názvy témat i obsah přenášených zpráv mají předem definovaný formát, všechny témata v systému EspHub používají jako kořenové téma `esp_hub/#`, obsah zpráv je pak ve formátu JSON. Formát JSON byl pro přenos obsahu zpráv zvolen z toho důvodu, aby bylo možné v jedné zprávě přenášet více datových hodnot. JSON lze také velmi snadno serializovat, deserializovat a validovat, jak na serveru, tak i na zařízení.

Následující tabulka 3 obsahuje tvar všech témat, které systém EspHub používá ke komunikaci a objasňuje význam jednotlivých témat.

Téma	**Význam tématu**
`esp_hub/device/ID/data`	Téma pro posílání zpráv mezi zařízením a serverem, kde ID je identifikátor zařízení.
`esp_hub/device/ID/telemetry`	Téma pro odesílání telemetrických dat ze zařízení, kde ID je identifikátor zařízení.
`esp_hub/device/hello`	Téma ve kterém zařízení oznamuje své připojení k serveru.
`esp_hub/device/ID/accept`	Téma ve kterém server potvrzuje zařízení, že bylo schváleno a může začít odesílat data. ID je identifikátor zařízení.
`esp_hub/device/ID/display`	Téma, kterým se přenáší obrazová data ze serveru na zařízení, kde ID je identifikátor zařízení.
`esp_hub/device/ID/cmd`	Téma pro přenos řídících zpráv ze serveru k zařízení, kde ID je identifikátor zařízení.
`esp_hub/api/request/ID`	Téma do kterého se posílají dotazy na MQTT API serveru. ID je unikátní identifikátor volání.
`esp_hub/api/response/ID`	Téma do kterého server publikuje odpovědi na API dotazy. ID je unikátní identifikátor volání.

Tabulka 3: Seznam použitých MQTT témat a jejich význam.

U témat `esp_hub/device/#` představuje hodnota ID unikátní identifikátor zařízení, v případě zařízení ESP jde o číslo odvozené z MAC adresy, které výrobce označuje jako ChipID. U témat sloužících pro získávání dat z MQTT API může být jako ID použita libovolná časově unikátní značka, například knihovna EspHubUnilib, která toto API využívá pracuje s identifikátory UUID dle RFC 4122 [4][5].

Formát obsahu zpráv není vždy fixní, příchozí zpráva nemusí vždy obsahovat všechny očekávané položky a naopak může obsahovat i položky nadbytečné, nicméně v rámci aplikace je definován doporučený formát, který by zařízení, případně server měly dodržovat. Následující tabulka 4 obsahuje seznam povinných a doporučených proměnných, které má zpráva obsahovat. U témat, které v tabulce nejsou uvedeny není doporučený formát specifikován a liší se dle konkrétního zařízení, či případu užití, například téma `esp_hub/api/request/ID` obsahuje zprávy ve zcela odlišném formátu, jejich syntaxe je popsána v kapitole 6.5.

Téma	Proměnná	Význam	Povinné
`esp_hub/device/ID/data`	type	identifikátor ability	ano
	value	hodnota ability	ano
	dvalue	hodnota chybového stavu	ne
`esp_hub/device/ID/telemetry`	rssi	síla signálu v síti WiFi	ne
	heap	množství dostupné paměti	ne
	cycles	doba běhu od zapnutí	ne
	local_ip	lokální IP adresa zařízení	ne
	mac	MAC adresa zařízení	ne
	voltage	napětí na procesoru	ne
	ssid	SSID WiFi přístupového bodu	ne
	hostname	hostname zařízení	ne
`esp_hub/device/hello`	name	název zařízení	ano
	id	unikátní identifikátor zařízení	ano
	ability	seznam poskytovaných funkcí	ano
	key	klíč pro symetrické šifrování	ne
`esp_hub/device/ID/accept`	ip	adresa MQTT brokeru	ano
	port	port MQTT brokeru	ano
	server_key	identifikační klíč serveru	ano
	name	jméno serveru	ne
`esp_hub/api/response/ID`	status	návratový status API volání hodnoty: `ok`, `error`, `nodata`	ano
	request_id	unikátní identifikátor volání	ano
	payload	výsledek API volání	ne

Tabulka 4: Seznam povinných a doporučených proměnných v obsahu MQTT zprávy.

Příklad obsahu zprávy, ve formátu JSON, zaslané ze zařízení na server v tématu `esp_hub/device/ID/data`:

```
{"type": "light", "value": "120", "dvalue": "0"}
```

Zpráva do tématu `esp_hub/device/hello` zaslaná zařízením na server a odpověď `esp_hub/device/ID/accept` zaslaná serverem na zařízení:

```
{"name": "HomeLightSensor", "id": "9503165", "ability": "['light','temp']"}
{"ip": "192.168.1.1", "port": 1883, "server_key": "a5-c7", "name": "HomeServer"}
```

5.3 Discovery protokol

Discovery protokol byl vytvořen za účelem automatického přidávání nových zařízení do správy serveru, s minimálním vynaloženým úsilím ze strany uživatele. Podmínky použití discovery protokolu jsou:

1. Zařízení musí být ve stejném síťovém segmentu jako server,
2. musí být k dispozici broadcastová IPv4 adresa,
3. zařízení musí mít k dispozic údaje pro připojení k WiFi síti.

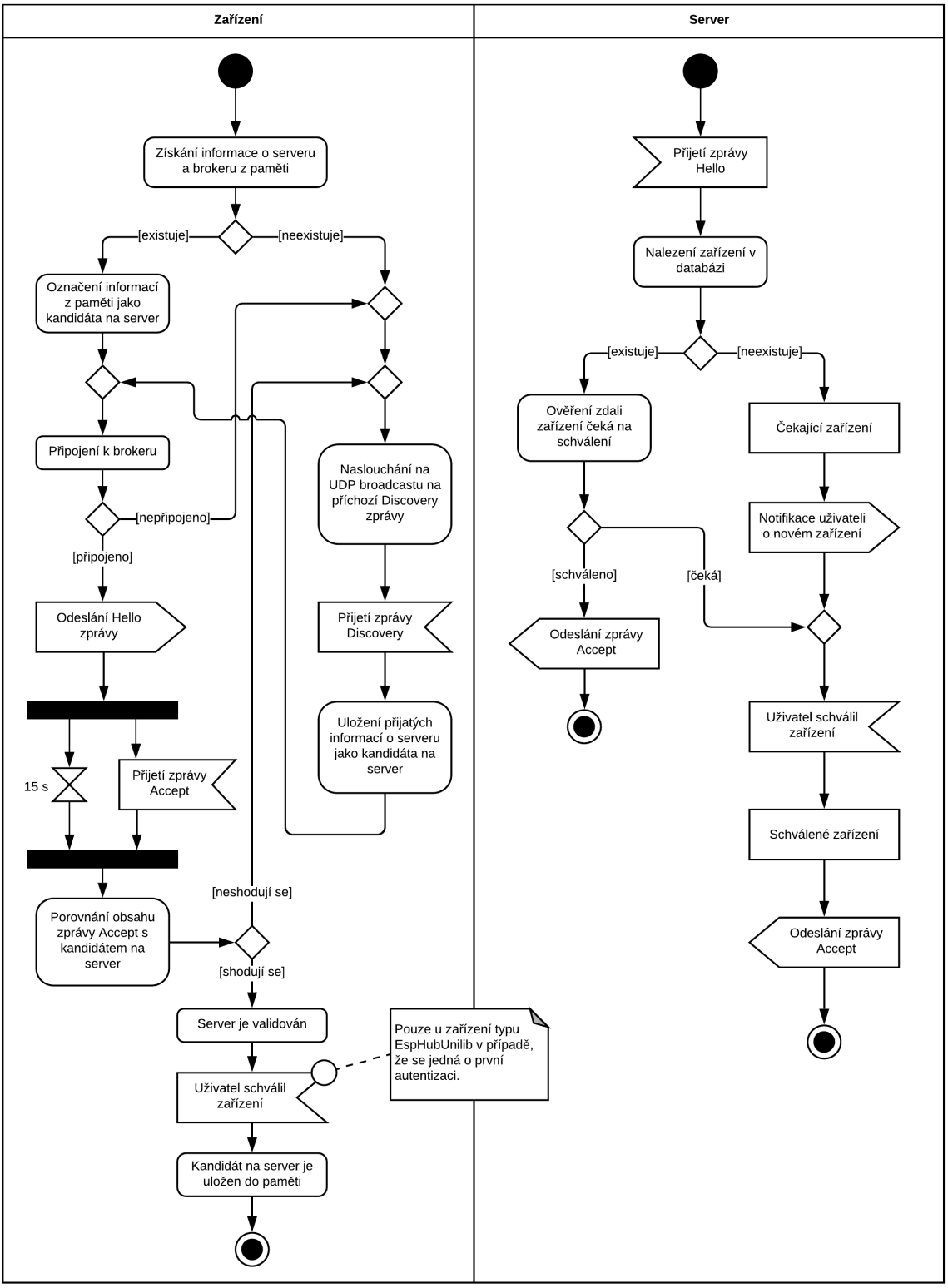
Discovery protokol pro svou funkci používá kombinaci protokolů UDP a MQTT. UDP protokol se využívá v prvotní fázi autentizace, serverech v pravidelných intervalech (ve výchozí stavu každých 5 s) rozesílá UDP zprávy na broadcastovou adresu lokální sítě, na port 11114, dále jen discovery zprávy. Obsah zprávy je ve formátu JSON a obsahuje informace o MQTT brokeru, ke kterému se má zařízení připojit. Zpráva obsahuje jméno serveru, IP adresu MQTT brokeru, port brokeru a identifikátor serveru, výsledná zpráva vypadá například takto:

```
{"ip": "192.168.1.1", "port": 1883, "server_key": "a5-c7", "name": "HomeServer"}
```

Hodnota `server_key` je unikátní identifikátor serveru, v současné verzi systému se jedná o UUID identifikátor vygenerovaný při prvním startu serveru, nicméně protokol byl navržen tak, aby mohl být použit i veřejný RSA klíč. Účelem zamýšleného RSA klíče by byla možnost provádět šifrovanou autentizaci, tato funkcionalita ovšem nebyla implementována kvůli, v současnosti, špatné podpoře šifrování na zařízeních ESP. UUID tedy slouží pouze, jako rozlišovací identifikátor serveru, zařízení pomocí něj identifikuje zdali se jedná o již známý server, ke kterému se v minulosti připojovalo, nebo jde o dosud neznámý server.

V pozdější fázi autentizace se používá protokol MQTT, konkrétně 2 témata `esp_hub/device/hello` a `esp_hub/device/ID/accept`, dále jen zprávy Hello a Accept. Zprávou Hello žádá zařízení server o navázání spojení a přidání do databáze zařízení, server odpovídá zprávou Accept.

Diagram na obrázku 12 zachycuje proces autentizace na straně zařízení a serveru. Zařízení po zapnutí ověří zda má v paměti uložené informace o serveru, pamětí je myšleno perzistentní úložiště dat. Pokud informace uložené má znamená to, že zařízení již v minulosti bylo spojeno s nějakým serverem a pokusí se k němu tedy opět připojit. Pošle serveru Hello zprávu a očekává potvrzení Accept, pokud Accept dorazí v definovaném časovém limitu 15 sekund, je server validován a zařízení může zahájit běžnou komunikaci. Na obrázku 13 je zachycen proces výměny zpráv mezi zařízením a serverem v případě, že zařízení již v minulosti navázalo spojení se serverem.

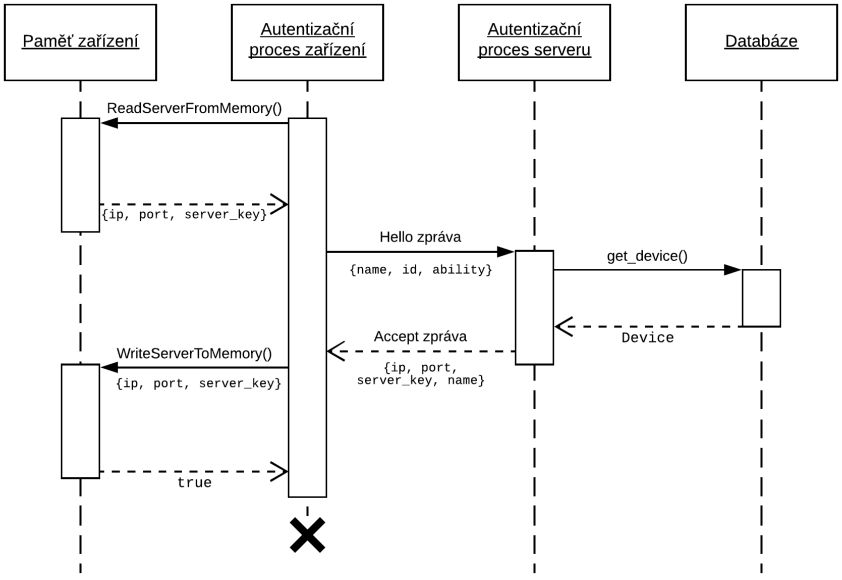


Obrázek 12: Diagram aktivity autentizace na zařízení a na serveru.

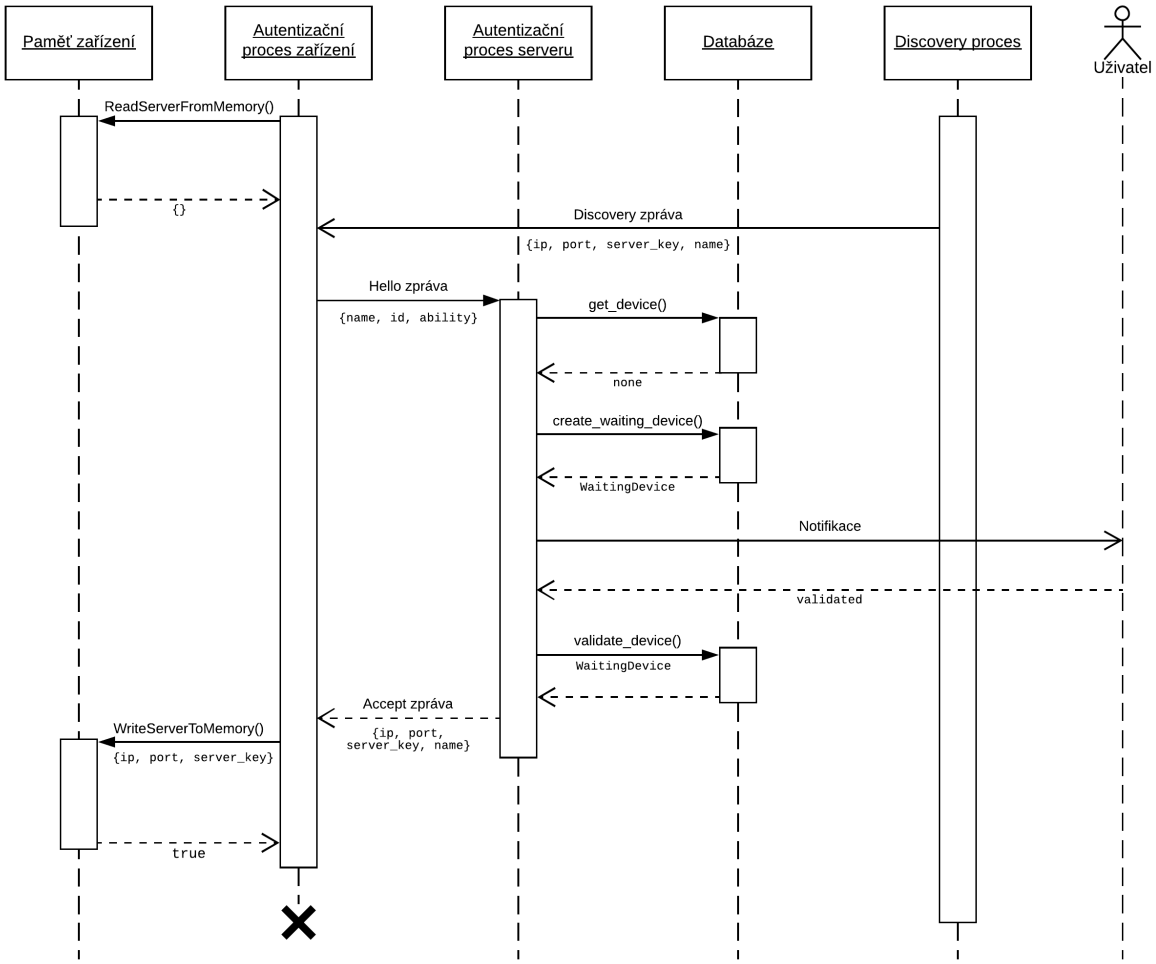
Nejsou-li v paměti žádné informace o serveru, nebo se s těmito údaji nedaří připojit k serveru, začne zařízení naslouchat na discovery zprávy, tento scénář je zobrazen na obrázku 12. Po přijetí discovery zprávy je zpráva rozparsována a údaje z ní jsou označeny za takzvaný kandidát na server. Zařízení se pokusí na kandidáta připojit, odešle Hello zprávu a čeká na potvrzení Accept. Dorazí-li potvrzení Accept v časovém limitu, označí zařízení kandidáta, jako validní server, uloží si informace o něm do perzistentní paměti a zahájí běžnou komunikaci. V případě zařízení ESP se validace serveru na straně zařízení provádí automaticky s první kandidát, který zašle zprávu Accept, je navázáno spojení, na zařízení EspHubUnilib ovšem musí uživatel ještě v konzoli potvrdit, že server skutečně akceptuje. Tento krok slouží jako bezpečnostní opatření a mohl by být implementován i u zařízení ESP, pokud by například disponovalo tlačítkem a displejem.

Na straně serveru probíhá autentizační proces následujícím způsobem: server naslouchá na zprávy Hello, ve chvíli kdy Hello zprávu zachytí vyhledá ve své databázi zařízení s odpovídajícím ID. Zařízení se může v databázi nacházet ve dvou stavech schválené a čekající (`validated` a `waiting`). Čekající zařízení je takové, od kterého už byla zachycena discovery zpráva, ale uživatel prozatím zařízení neschválil, schválené zařízení je takové, které už bylo uživatelem validováno. Jestliže je zařízení s daným ID nalezeno v databázi a je schváleno, server bez prodlení odesílá zprávu Accept. U zařízení ve stavu čekající není zpráva Accept odeslána dokud uživatel zařízení přes webové rozhraní nevaliduje. V případě, že zařízení v databázi není nalezeno, vytvoří server nové čekající zařízení a notifikuje uživatele, že je nové zařízení k dispozici.

Uživatel při validaci může ještě upravit vlastnosti zařízení, změnit jeho jméno a upravit vstupní a výstupní vlastnosti, které zařízení poskytuje.



Obrázek 13: Sekvenční diagram autentizace zařízení v případě kdy zařízení má uloženy informace o serveru a server je validní.



Obrázek 14: Sekvenční diagram autentizace zařízení v případě kdy zařízení nemá uloženy informace o serveru a inicializuje naslouchání na discovery zprávy.

5.3.1 Šifrovaná autentizace

Přestože tato varianta protokolu není v současné verzi systému plně implementována, systém je na případné rozšíření připraven. Proces autentizace by byl obdobný, jako u nešifrované verze: Server při prvotním spuštění vygeneruje pár RSA klíčů privátní a veřejný. Veřejný klíč je společně s dalšími údaji pravidelně rozesílán broadcastovou UDP zprávou v poli `server_key`. Zařízení si při prvotním spuštění vytvoří AES klíč pro symetrické šifrování a tento klíč si uloží do své flash paměti, klíč bude používán opakovaně při každém připojování k serveru. Ve chvíli kdy zařízení zachytí discovery zprávu ze serveru, uloží si příchozí údaje, jako kandidáta na server. Zařízení použije příchozí `server_key` pro zašifrování svého AES klíče. Tento zašifrovaný klíč následně zašle v poli `key` v Hello zprávě serveru. Poté co uživatel zařízení validuje, uloží si server údaje o zařízení do databáze včetně zmiňovaného AES klíče, který rozšifruje pomocí svého privátního klíče a odpoví zařízení zprávou Accept, kterou ovšem celou zašifruje AES klíčem, který od

zařízení obdržel. Zařízení přijme zašifrovanou zprávu Accept a pokusí se jí opět zrekonstruovat svou kopií AES klíče, pokud údaje ve zprávě souhlasí s kandidátem na server, uloží si údaje o serveru do flash paměti.

Po úspěšné autentizaci mají server i zařízení sdílený AES klíč, který mohou využít pro symetrické šifrování všech zpráv, které si mezi sebou vyměňují.

Proces šifrované autentizace zajišťuje, že se zařízení omylem nepřipojí k jinému serveru, který by předstíral, že je validním serverem. Takový server by totiž neměl k dispozicí správný privátní klíč a nemohl tak získat ze zprávy Hello AES klíč zařízení.

Je také vyloučeno podvržení zařízení, server nezašle zprávu Accept zařízení, která již má ve své databázi uložené a obsahuje jiný AES klíč. Pokud by chtěl uživatel použít stejné zařízení, ale s nově vygenerovaným AES klíčem, musí zařízení nejdříve smazat z databáze spravovaných zařízení.

6 EspHubServer

Komponenta EspHubServer slouží jako centrální prvek systému, hlavní doménou komponenty je komunikace s databází a MQTT brokerem. EspHubServer registruje všechna MQTT témata, do kterých zařízení posílají svá data (úplný seznam je k dispozici v tabulce 3) a následně naslouchá na všechny příchozí zprávy v těchto tématech. Příchozí zprávy jsou validovány a uloženy do databáze, EspHubServer server poté přehled o nasbíraných data zprostředkovává uživateli skrze webové rozhraní, ve kterém může uživatel zkontrolovat aktuální stav zařízení a prohlédnout si grafy s nasbíraných dat.

Komponentu EspHubServer lze rozdělit na několik funkčních komponent, jednotlivé komponenty mohou pracovat téměř samostatně a není ani nutné, aby byly spuštěny na stejném počítači, nebo ve stejné síti, jedinou podmínkou je přístup ke společné databázi. Uživatel si při spuštění EspHubServeru, pomocí spouštěcí utility vybere, které komponenty si přeje aktuálně spustit. Jedná se o tyto komponenty:

Web interface Jedná se o webové rozhraní, které uživateli zprostředkovává správu zařízení a zobrazení nasbíraných dat.

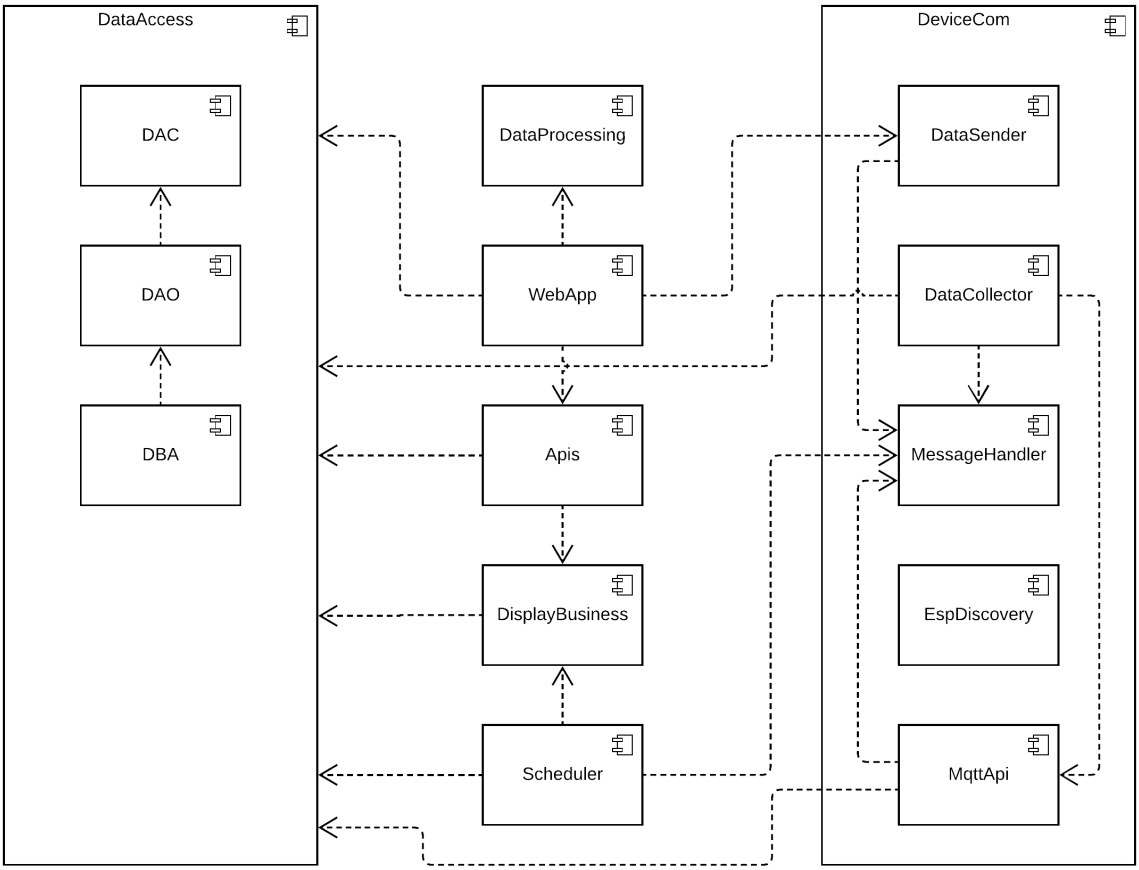
Data collection Tato část zajišťuje sběr dat z brokeru a vyřizování požadavků na MQTT API. Přijatá data validuje a ukládá do databáze.

Device discovery Tato komponenta rozesílá na broadcastovou adresu lokální sítě discovery zprávy pro zařízení. Zařízení díky se díky těmto zprávám dokáží automaticky objevit server a připojit se k němu.

Scheduler Neboli plánovač úloh, zajišťuje běh naplánovaných operací, jako například odesílání dat na displeje.

6.1 Moduly systému

EspHubServer je tvořen celkem 9 moduly napsanými v jazyce Python a několika pomocnými funkcemi, tyto moduly mezi sebou různým způsobem kooperují a dohromady tvoří jednotlivé funkce a komponenty systému. V následujících odstavcích bude popsán účel a vysvětlena funkce jednotlivých modulů. Diagram na obrázku 15 zachycuje vysokoúrovňový pohled na moduly systému, každý z modulů je tvořen nejméně jednou funkcí nebo třídou, některé moduly jako například WebApi jsou tvořeny mnoha třídami. Z diagramu byly některé méně podstatné vazby mezi moduly odstraněny z důvodu přehlednosti, například moduly Config a Tools mají vazbu na téměř všechny moduly v systému.



Obrázek 15: Diagram modulů systému EspHubServer.

DataAccess Modul DataAccess je tvořen především 3 submoduly: **DAO** (Data Access Objects), **DBA** (Database Access), **DAC** (Database Connection). DAO obsahuje třídy pro objektově relační mapování databázových entit. DBA slouží pro zjednodušení přístupu k databázi, ostatním modulům poskytuje funkce pro operace s daty, vkládání, vyhledávání a modifikaci záznamů. DAC obstarává spojení s databází a inicializuje objektově relační mapování.

DeviceCom Tato část zajišťuje komunikaci se zařízeními a to především pomocí MQTT protokolu, nejvýznamnějším submodulem je **MessageHandler**, ten obaluje externí modul `paho.mqtt.client`, který je implementací MQTT klienta pro jazyk Python. MessageHandler zajišťuje navázání spojení s brokerem a udržuje ho stále aktivní, registruje témata a callback metody pro zpracování příchozích zpráv, také poskytuje surové rozhraní pro publikaci zpráv. Publikační funkce MessageHandleru používá submodul **DataSenter**, který pouze přidává k odesílaným datům správná témata.

Dalším důležitým submodulem je **DataCollector** ten obsahuje pro každé téma separátní callback, který zpracovává příchozí zprávy, také obsahuje pomocné metody pro parsování zpráv. Tento submodul se taky například stará o odesílání Hello zpráv, tedy odpovědí na volání Accept.

EspDiscovery distribuuje zprávy discovery, díky tomuto submodulu mohou zařízení automaticky detekovat přítomnost serveru v lokální síti. Úkolem modulu **MqttApi** je odpovídat na dotazy vznesené přes MQTT aplikační rozhraní, hlouběji popsané v kapitole 6.5.

WebApp Tento modul[6]zprostředkovává webové uživatelské rozhraní, pomocí něhož uživatel spravuje zařízení, prohlíží statistiky a nastavuje server. Funkčnost tohoto modulu je detailněji popsána v kapitole 6.3.

Apis Vytváří HTTP REST rozhraní pro přístup k datům. Modul Apis je často používán v kombinaci s modulem WebApp, mnohá volání z webové aplikace směřují právě do modulu Apis.

Scheduler Neboli plánovač úloh, byl vytvořen tak, aby mohl zpracovávat libovolné pravidelné úlohy, samotné úlohy běží v paralelních procesech a neblokují tedy zbytek aplikace ani sebe navzájem. Ačkoliv je plánovač implementován tak, aby mohl zpracovávat teoreticky libovolný typ úloh, v současné verzi systému slouží pouze pro pravidelné odesílání obsahu pro displeje připojené k zařízením.

DisplayBusiness Používá se pro zpracování dat a renderování obrazu pro displeje připojené k zařízením, více informací o fungování modulu je v kapitole 6.4.

DataProcessing Modul je používán webovým rozhraním pro zpracování dat určených pro grafy. DataProcessing pracuje se statistickou knihovnou Pandas a dokáže pomocí ní provádět time series operace nad daty určenými pro grafy.

Config Spravuje konfigurační soubory a poskytuje ostatním modulům přístup k těmto konfiguracím.

Tools Obsahuje pomocné funkce používané napříč aplikací, především se jedná o modul **Log**, který je používán pro výpis provozních informací, ty jsou vypisovány do konzole a zároveň ukládány do souboru. Log umožňuje nastavit úroveň logování, uživatel tak může zvolit zda chce zobrazovat pouze chybová hlášení, nebo i informační a ladící výpisy.

Následující kapitoly budou věnovány hlubšímu pohledu na některé podstatné funkce systému. Funkce jsou vždy kooperací několika modulů systému.

[6]Modul z historických důvodů v systému často vystupuje pouze pod názvem **main**.

6.2 Datová vrstva

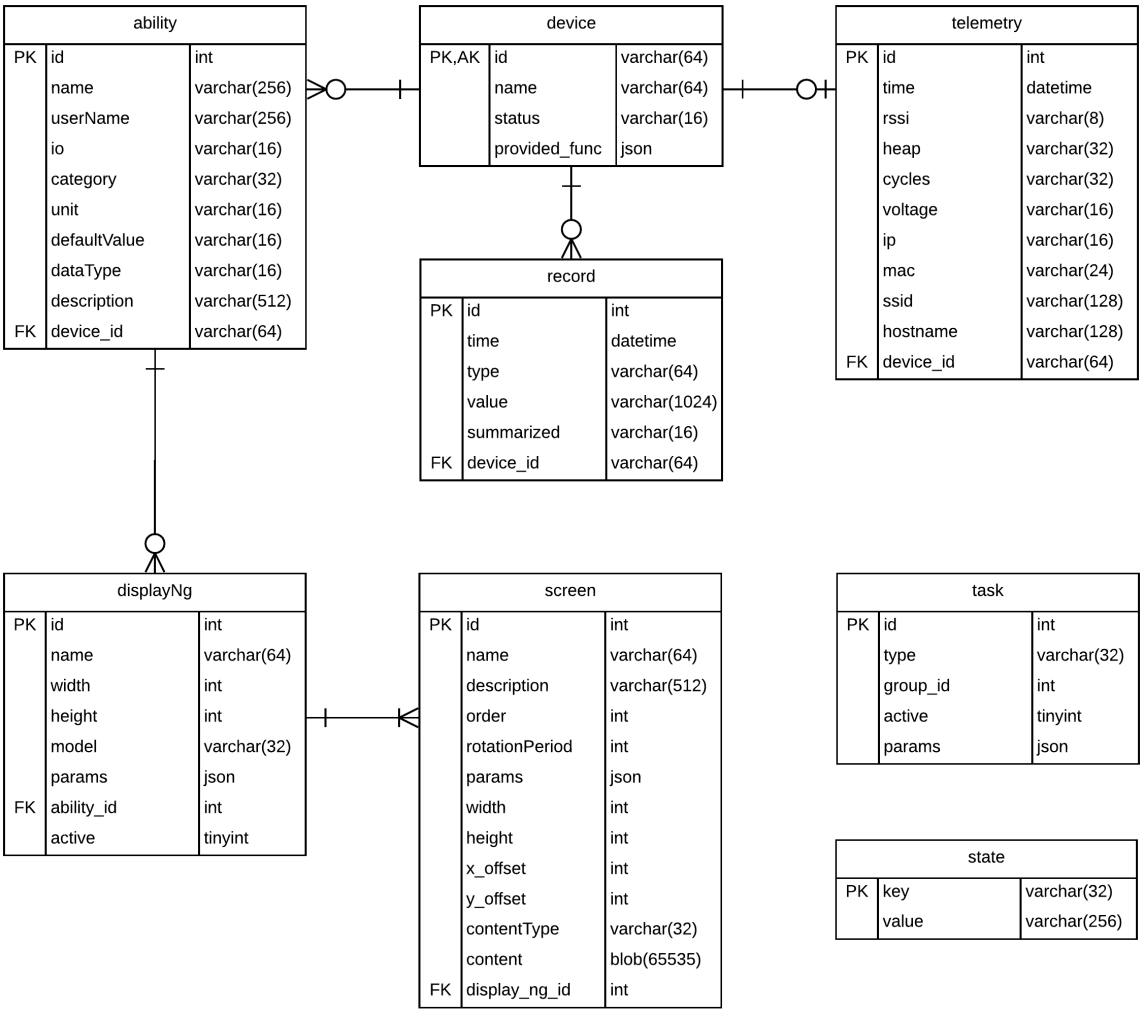
Při návrhu aplikace jsem zvažoval různé možnosti, jak datovou vrstvu implementovat, původní myšlenka byla udržet databázovou vrstvu minimalistickou a redukovat nutnost instalace databázového serveru. Proto jsem jako primární DBMS (database management system) zvolil SQLite, ten totiž celou databázi reprezentuje jako jediný soubor. Tento přístup přináší několik výhod: systém je snadno přenositelný, lze jej snadno zálohovat, není nutné instalovat a konfigurovat databázový server. Pochopitelně jsou zde i nevýhody, těmi hlavními jsou pomalé vkládání a vyhledávání dat, což se v pozdějších fázích vývoje ukázalo, jako kritické a vyvstala potřeba použít DBMS s větší výkonností. To jestli je lepší použít SQLite, nebo například MySQL závisí na tom, jak se uživatel rozhodne systém používat, pokud sbírá data z jednotek zařízení na lehkém Linuxovém počítači, kupříkladu RaspberryPi, použití databáze SQLite nebude uživatele nijak omezovat a naopak sníží nároky na výkon počítače. Pokud má uživatel k dispozici výkonnější server a sbírá data z desítek zařízení, je mnohem vhodnější použít výkonnější databázi. Rozhodl jsem se tedy toto rozhodnutí ponechat na uživateli a implementoval systém tak, aby byl přenositelný napříč databázovými systémy.

Prvotní plán implementace vlastního objektově relačního mapování (ORM) byl také rychle opuštěn, jednak z důvodu přenositelnosti napříč databázovými systémy a také z toho důvodu, že implementace mnoha metod pro přístup k datům a zabírala neadekvátní množství času, vytvářením mnoha funkcí pro mapování relačních struktur databáze na objekty se stával kód nepřehledný. Právě z těchto důvodů jsem se rozhodl použít framework pro objektově relační mapování, pro jazyk Python je k dispozici několik různých frameworků, ale zdaleka nejpopulárnější a nejrozšířenější je framework SQLAlchemy [41], ten jsem také po důkladném otestování použil.

SQLAlchemy má přímou podporu pro nejrozšířenější databázové systémy [42] (Microsoft SQL Server, MySQL, Oracle, atd.) a umožňuje mezi těmito systému přepínat pouze záměnou přihlašovacího řetězce, splňuje tak můj požadavek na přenositelnost napříč databázovými systémy.

Objekty pro relační mapování byly vytvořeny implementací rozhraní SQLAlchemy, tyto třídy jsou obsaženy v modulu DAO, obsahují jak jednotlivé datové položky, tak relační vazby na ostatní entity. CRUD operace s těmito entitami zajišťuje modul DBA, který pro ostatní moduly slouží, jako rozhraní pro práci s databází. Modul DAC pak zajišťuje správu spojení s databází a inicializuje a ukončuje jednotlivé transakce.

Na obrázku 16 je úplný relační model databáze. Každé zařízení (tabulka `device`) může mít více schopností (tabulka `ability`) a datových záznamů (tabulka `record`). Zařízení má vazby jedna ku jedné s telemetrií (tabulka `telemetry`), zařízení přijímá telemetrii kontinuálně a nemá význam všechna příchozí telemetrie ukládat, databáze tedy vždy obsahuje pouze poslední přijatou telemetrickou zprávy. Jednou ze schopností zařízení může být připojený displej, ten je reprezentován tabulkami `displayNg` a `screen`, který reprezentuje virtuální obrazovky. Tabulky `task` a `state`, nemají přímou vazbu na ostatní entity a jsou využívány plánovačem úloh.



Obrázek 16: Diagram entit v databázi.

6.2.1 TimeSeries databáze

V průběhu implementace systému vyvstala otázka, jak efektivně odstraňovat staré neaktuální záznamy z databáze, na základě výzkumu se ukázalo že nejvhodnějším řešením pro uchovávání tohoto typu dat, tedy časově vázané záznamy o velké kvantitě, jsou takzvané time series databáze.

Time series data jsou sekvenčně dodávaná data s vazbou na konkrétní čas pořízení, ukládaná v databází v časovém pořadí, typickým příkladem takovýchto dat jsou záznamy z různých měřících zařízení, které jsou dodávány kontinuálně s vazbou na konkrétní čas [46][47]. Existují databázové systémy, které se ukládání obrovského množství dat v tomto formátu specializují a umožňují nad těmito daty provádět specifické operace efektivněji. Specializované databázové systému dokáží datům přidat automaticky dobu expirace, po které budou data smazána, rovněž umožňují nad daty provádět sumarizace a statistické operace. Například je mnohdy zbytečné

ukládat data se vzorkovací frekvencí 1 s po dobu několika let, při pohledu na historická data postačí většinou rozlišení v řádu hodin, týdnů, nebo měsíců, time series databáze dokáží data tímto způsobem zpětně automaticky sumarizovat. Zástupci time series databází jsou například eXtremeDB, nebo populární InfluxDB [48].

Vzhledem k tomu, že systém EspHub pracuje právě s tímto typem dat operuje, zabýval jsem se myšlenkou integrace některé z time series databází. Tyto databáze jsou ovšem, jako zástupci NoSQL databází [49], nekompatibilní s tradičními nástroji pro objektově relační mapování, navržený model databázové vrstvy by tak bylo nutné z větší části přepracovat. Pro tyto úpravy nicméně již nezbyl v projektu prostor a integrace těchto nástrojů tak představuje příležitost pro další případný rozvoj systému.

6.3 Webové rozhraní

Webové rozhraní bylo vytvořeno tak aby poskytovalo uživatelsky přívětivý přístup k funkcím a správě celého systému. Schéma na obrázku 17 znázorňuje celkovou strukturu webového portálu.

Seznam zařízení, na obrázku 19, je zároveň domovskou obrazovkou, zobrazuje seznam všech zařízení v systému a indikuje jejich stav tedy online/offline. Stav je odvozen od telemetrických zpráv, pokud systém po dobu více než 30 s nepřijme od zařízení telemetrickou zprávu, označí zařízení jako offline. Po kliknutí na položku seznamu se zobrazí detail zařízení, na obrázku 20, a v něm pak uživatel může vidět nejaktuálnější přijatá data ze zařízení, grafy naměřených hodnot, ovládací prvky a telemetrické údaje, struktura stránky je naznačena na obrázku 18.

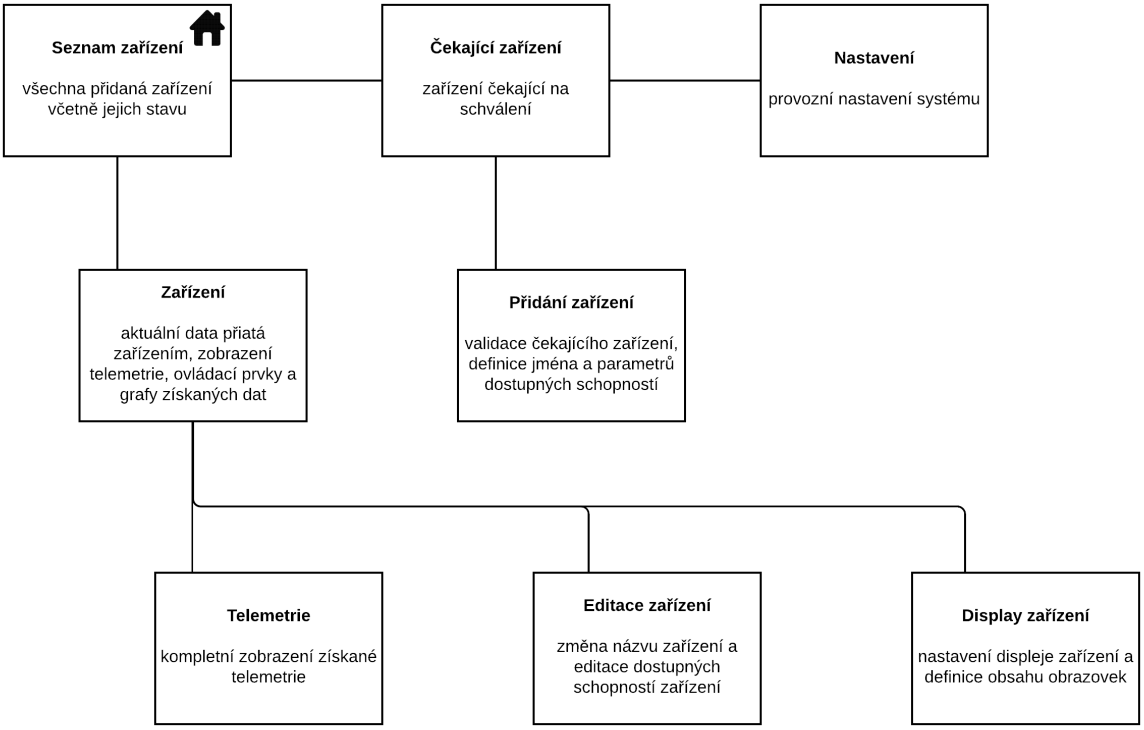
U zařízení která poskytují data jsou ve spodní části stránky zobrazeny grafy vytvořené na základě přijatých dat, grafy jsou konstruovány pouze tehdy, lze li přijatá data převést do číselného formátu. U grafu si uživatel může nastavit časový rozsah zobrazených dat, vzorkování dat (po minutách, hodinách, dnech, nebo týdnech) a typ grafu (spojnicový, spojnicový s výplní, sloupcový). Zobrazený graf je interaktivní, uživatel může data přibližovat, pohybovat se v nich a uložit graf jako obrázek do svého počítače. Na obrázku 21 je ukázka grafu zobrazujícího vývoj teploty v definovaném časovém období.

6.3.1 Použité technologie

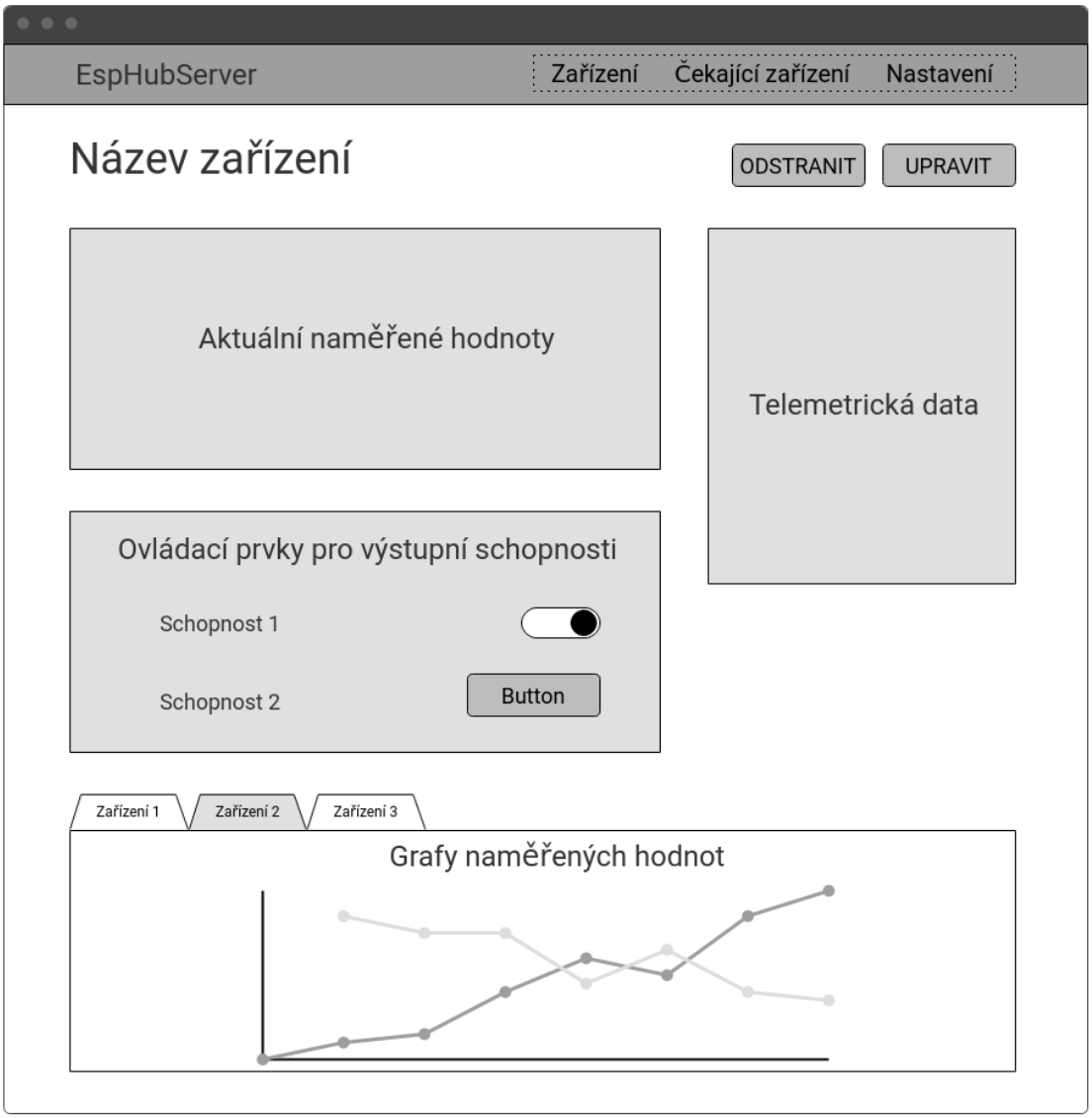
Webový portál byl vytvořen pomocí frameworku Django, jedná se o jeden z nejpopulárnější nástrojů pro tvorbu webových aplikací v jazyce Python. Aplikace pro svůj běh ve výchozím nastavení používá integrovaný webový server frameworku Django, ten sice nebyl navržen pro produkční nasazení, ale pro provoz jednouživatelské aplikace postačuje. Vzhled aplikace je definován CSS šablonou Materialize ve verzi 0.98.0, ta se snaží dodržovat designový jazyk Google Material Design a připomíná tak vzhledem aplikace pro platformu Android. Pro tvorbu interaktivních prvků v uživatelském rozhraní byl ve velké míře využíván JavaScript v kombinaci s různými knihovnami, především pak JQuery.

Moduly webového rozhraní zahrnují kromě samotného uživatelského portálu ještě REST API rozhraní, to obsahuje, v současné verzi, pouze několik základních metod pro získávání a modifikaci záznamů, je používáno skripty v portálu pro načítání dat v reálném čase.

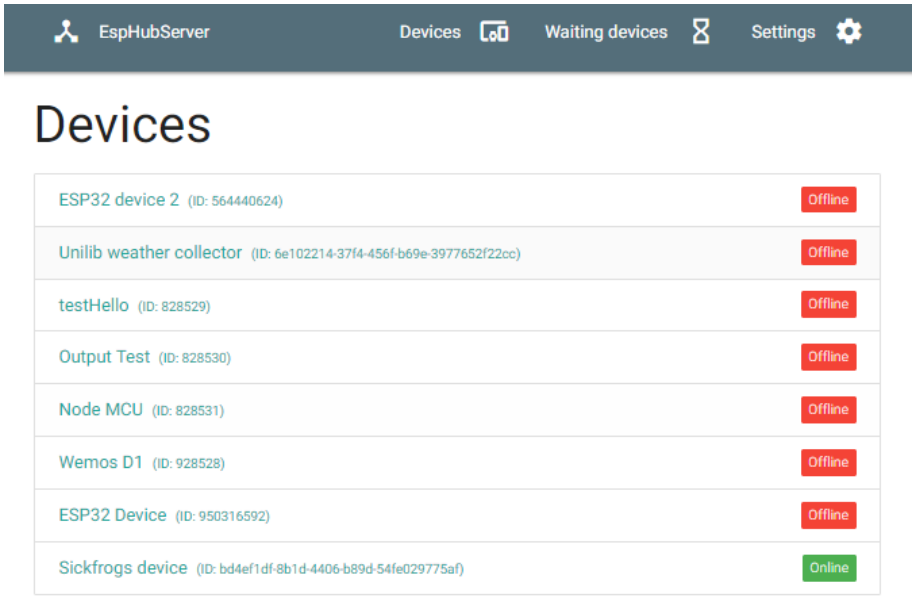
Pro změnu vzorkování časových údajů u grafů slouží modul DataProcessing, který používá statistickou knihovnu Pandas, ta dokáže mimo jiné provádět sumarizace nad poskytnutými time series daty.



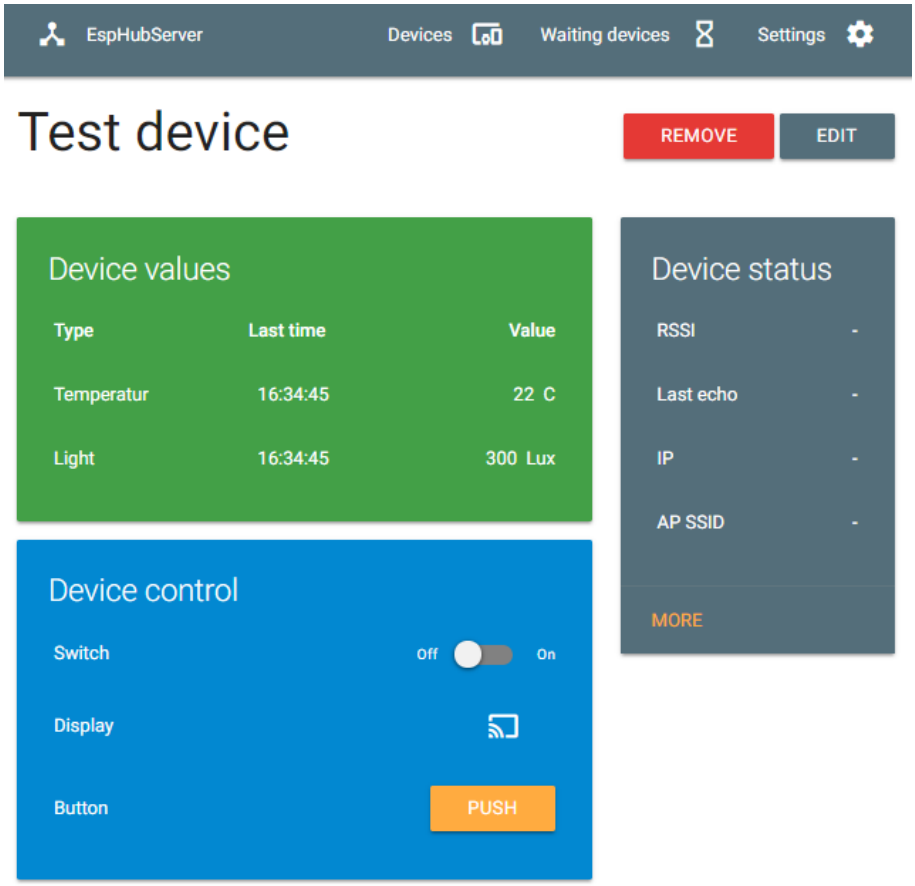
Obrázek 17: Mapa webového rozhraní.



Obrázek 18: Struktura stránky s detailem zařízení.



Obrázek 19: Úvodní stránka se seznamem dostupných zařízení.



Obrázek 20: Stránka s detaily o zařízení.

Obrázek 21: Ukázka grafu v detailu zařízení.

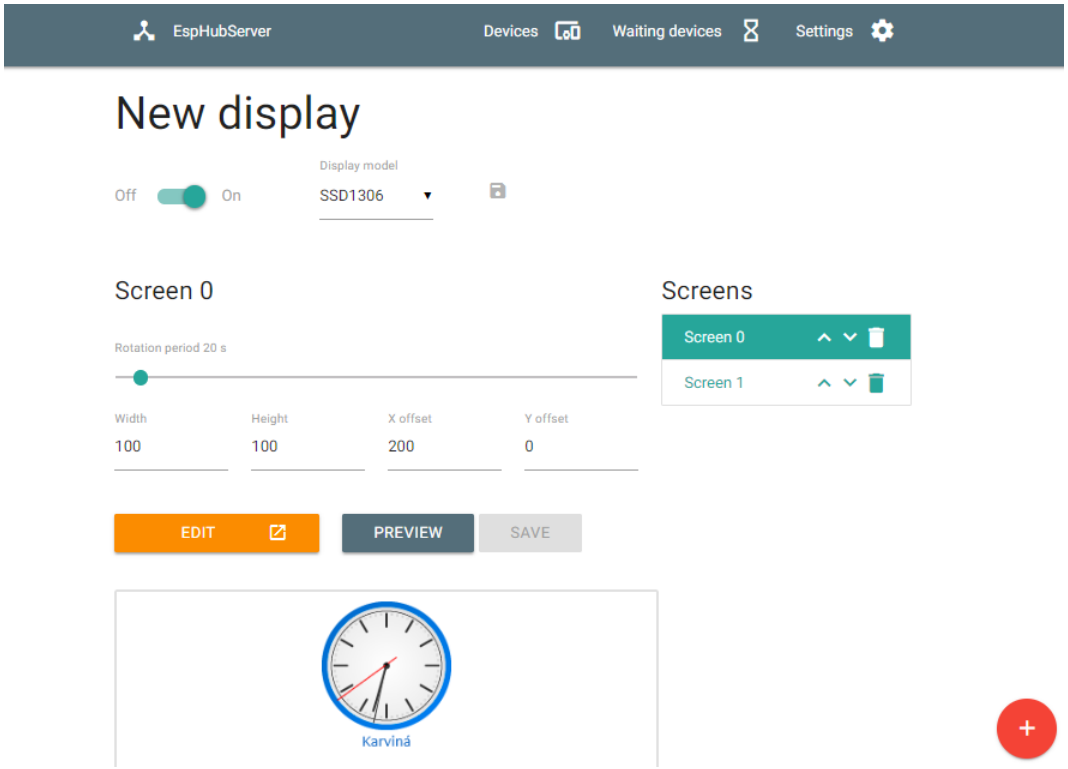
6.4 Generátor obsahu displeje

Zařízení k sobě mohou mít připojený displej, jako jedno z volitelných výstupních zařízení, pokud se jedná o podporovaný displej, uživatel může definovat přes webové rozhraní obsah zobrazený na tomto displeji. Při návrhu bylo zvažováno několik různých možností, jak uživateli tuto funkcionalitu přes webové rozhraní zpřístupnit, jednou z myšlenek byly předdefinované grafy, ve kterých by uživatel pouze zvolil zobrazovanou veličinu, například hodnotu naměřenou jedním ze zařízení. Tento přístup se však ukázal být velmi omezující a nevyužíval plný potenciál displejů. Daleko plnohodnotnější metodou by bylo uživateli zpřístupnit jakýsi WYSIWY[7] editor obsahu displeje, vývoj takového editoru je ovšem nad rámec této práce. Výsledné rozhraní je tedy kompromisem mezi uživatelskou přívětivostí a časovou náročností implementace.

Uživatel definuje obsah obdobně jako webovou stránku, má k dispozici editor se zvýrazněním syntaxe a může používat libovolnou kombinaci jazyků HTML, CSS a JS. Uživatel tímto způsobem může definovat více obrazovek, které se budou na displeji střídat, může definovat periodu střídání i pořadí, ve kterém se obrazovky budou vykreslovat. V průběhu vývoje si uživatel může zobrazit náhled toho, jak výsledná obrazovka bude vypadat. Systém před zobrazením na displeji vložený kód na pozadí vyrenderuje pomocí webového prohlížeče Google Chrome, jako webovou stránku a pořídí snímek výsledné stránky. Snímek je oříznut podle parametrů definovaných

[7] „What you see is what you get“ v češtině „co vidíš, to dostaneš“ jsou editory, ve kterých obsah editovaný uživatelem má stejnou podobu jako výsledný obsah.

uživatelem, převeden do formátu vhodného pro konkrétní typ displeje a odeslán přes protokol MQTT na displej. Na obrázku 22 je vidět obrazovka z nastavením displeje a jeho obrazovek, s již vykresleným náhledem v podobě analogových hodin.



Obrázek 22: Stránka s nastavením displeje a jeho obrazovek.

6.5 MQTT aplikační rozhraní

MQTT aplikační rozhraní bylo vytvořeno, jako doplněk pro komunikaci mezi serverem a zařízeními, které nemají přístup k REST API rozhraní a komunikují pouze přes MQTT protokol. Rozhraní slouží především pro komunikaci mezi serverem a knihovnou EspHubUnilib popsanou v kapitole 8. V současné verzi je definováno pouze několik volání pro získávání dat ze serveru, například metoda `get_device_id`, která na základně zadaného jména vyhledá ID zařízení, nebo `get_device_ip`, která na základě ID zařízení vyhledá jeho poslední známou IP adresu.

Pro zasílání dotazů a odpovědí slouží vyhrazená témata `esp_hub/api/request/ID` a `esp_hub/api/response/ID`, zmíněná již v kapitole 5.2.1. Klient zašle do tématu `esp_hub/api/request/ID` dotaz ve specifickém formátu, který vždy zahrnuje příkaz následovaný volitelnými argumenty, jejich počet a tvar závisí na příkazu, příkladem takového volání je: `get_device_id device_name="Home temperature device"`

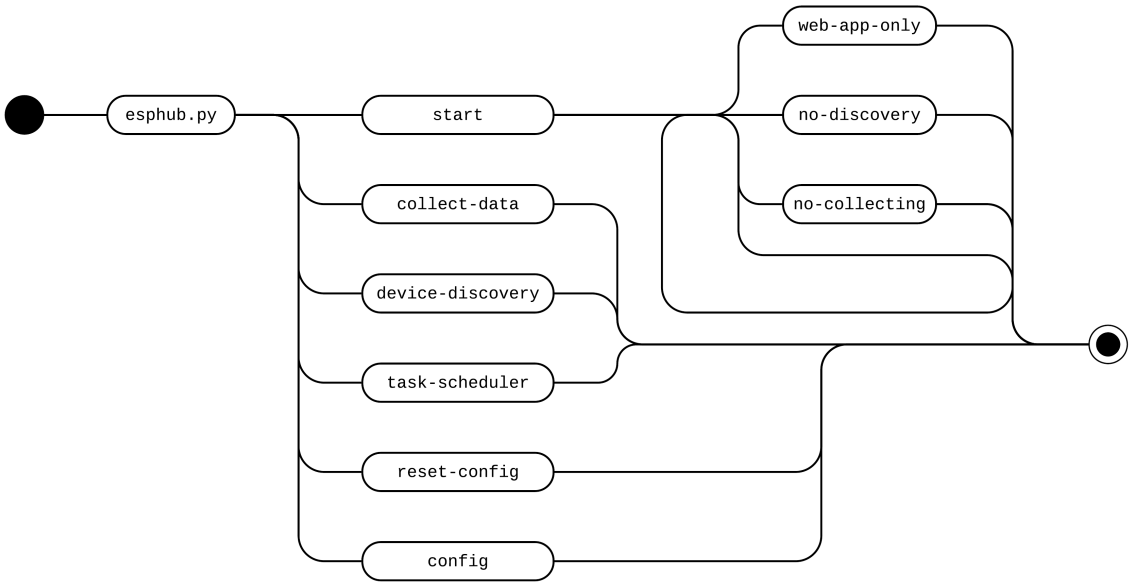
Odpověď je uživateli vrácena ve formátu JSON do tématu `esp_hub/api/response/ID`, zpráva vždy obsahuje položky `status`, `payload` a `request_id`. Status nabývá hodnot `ok`, `error`

a `nodata`, podle toho jestli byl dotaz validní a podařilo, nebo nepodařilo se nalézt nějaká data odpovídající dotazu, payload je pak samotný obsah odpovědi, který se liší v závislosti na dotazu. Položka `request_id` je pouze kopií položky ID v názvu tématu, toto ID je unikátní identifikátor zaslaný odesilatelem, aby bylo možné jednoznačně rozeznat, která odpověď patří ke kterému dotazu.

6.6 CLI rozhraní

CLI rozhraní je nástroj, který byl vytvořen pro snadnější spouštění systému EspHubServer, je napsán v Pythonu a používá knihovnu Click. Tato knihovna pomáhá při tvorbě interaktivních skriptů v příkazové řádce, dokáže zpracovávat argumenty předané z příkazové řádky, validovat jejich obsah a automaticky vygenerovat nápovědu, pokud uživatel zadá neplatný argument [50].

Na obrázku 23 je vidět diagram syntaxe tohoto nástroje, příkaz `start` aktivuje všechny komponenty systému, uživatel si může volitelně některé z nich vypnout. Nebo je možné zapnout pouze jednu z komponent `collect-data`, `device-discovery`, nebo `task-scheduler`. Ostatní příkazy slouží pro modifikaci, nebo resetování konfigurace serveru.



Obrázek 23: Syntaxe spouštěcího skriptu pro systém EspHubServer.

7 Knihovna EspHubLib

EspHubLib je knihovna pro platformu Arduino, která umožňuje uživateli navázat komunikaci s EspHubServerem. Poskytuje funkcionalitu pro automatické nalezení serveru pomocí discovery protokolu a uložení informací o serveru do paměti zařízení. Dále pak zpřístupňuje uživateli rozhraní, díky kterému může ze svého programu odesílat data na server a také data přijímat. Zajišťuje tak navázání spojení s MQTT brokerem, převedení zprávy do formátu JSON a odeslání se správným tématem, také automaticky zasílá serveru informace o stavu zařízení.

Vzhledem k tomu, že Arduino Core SDK pro ESP32 se mírně liší od toho určeného pro ESP8266 není knihovna mezi verzemi kompatibilní. Ke změnám došlo v metodách pro ovládání WiFi připojení mikropočítače, některých funkcích pro získávání vnitřního stavu modulu a také byla předefinována knihovna sloužící k přístupu do flash paměti. Z tohoto důvodu byly vytvořeny dvě implementace knihovny EspHubLib, pro každou z verzí mikropočítače, rozhraní knihoven jsou ovšem velmi podobná.

7.1 Funkce knihovny EspHubLib

Knihovna dává uživateli k dispozici několik metod, pomocí kterých může navázat spojení se serverem a následně odesílat data. Postup práce s knihovnou je následující: uživatel vytvoří instanci knihovny a předá jako parametr jméno zařízení, zajistí připojení zařízení k WiFi síti, pomocí metody `setAbilities` nastaví ability, které bude zařízení poskytovat. Volitelně může manuálně nastavit server, pomocí metody `setServer` a callback funkci pro zpracovávání příchozích zpráv, metodou `setCallback`. Následně je nutné zavolat metodu `begin`, která dá knihovně pokyn k navázání komunikace se serverem, pokud v předchozích krocích nenastavil uživatel manuálně server, započne zde proces automatického hledání serveru. Metoda `begin` blokuje chod programu do chvíle než se podaří připojit k serveru, v Arduino programech je tedy vhodné celou tuto část umístit do funkce `setup`.

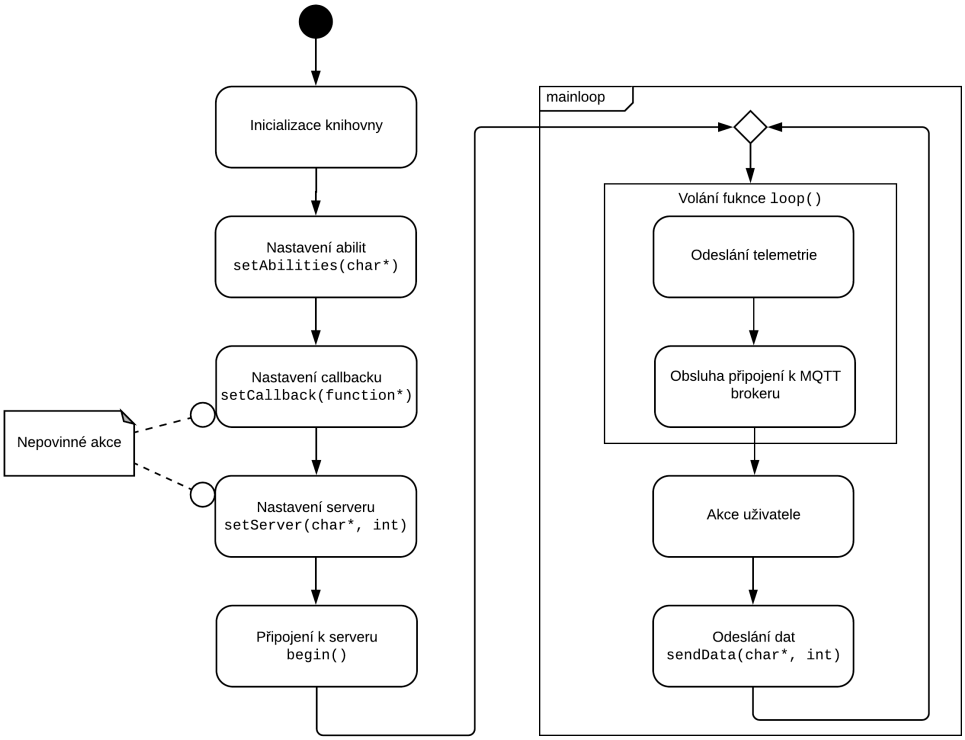
Ve chvíli kdy se podaří navázat komunikaci se serverem a zařízení začne vykonávat svůj mainloop[8], může uživatel odesílat data pomocí metod `sendData` a `sendJson`. Metoda `sendData` akceptuje, jako parametr typ zprávy, tedy jednu z abilit, kterou uživatel definoval při inicializaci a obsah zprávy, kterým může být řetězec, celé číslo, nebo desetinné číslo. Metoda `sendData` je preferovaný způsobem odesílání dat, ve speciálních případech lze použít metodu `sendJson`, kterou jinak interně používá metoda `sendData`.

V hlavní smyčce je uživatel povinen pravidelně volat metodu `loop`, která zajišťuje pravidelnou obsluhu interních procesů knihovny, jako například odesílání telemetrie, obsluha fronty zpráv a udržování spojení s MQTT brokerem. Důvodem tohoto řešení je absence podpory paralelizmu na platformě Arduino, přestože ESP32 má k dispozici více jader. Pokud uživatel tuto

[8] Hlavní smyčka programu, ve které se opakovaně provádějí akce definované uživatelem.

metodu nezavolá alespoň jednou za 30 s dojde u zařízení ke změně stavu na offline a zařízení se může eventuálně nedobrovolně odpojit od MQTT brokeru.

Na obrázku 24 je vizualizována aktivita inicializace knihovny a její následné používání uživatelem.



Obrázek 24: Vizualizace aktivit knihovny EspHubLib.

7.2 Přihlášení k bezdrátové síti

Jedním z mála rozdílů mezi verzí knihovny pro ESP8266 a ES32 je způsob, jakým uživatel předává přihlašovací údaje k lokální WiFi síti. V počátcích návrhu nebylo cílem knihovny řešit proces připojení zařízení k bezdrátové síti, knihovna počítala s tím, že uživatel zajistí připojení manuálně, pomocí nativních metod mikropočítače, nebo nějakou alternativní knihovnou. V ukázkovém příkladu pro modul ESP8266 používám knihovnu WiFiManager, která dokáže vytvořit captive portál, do kterého uživatel zadá přihlašovací údaje ke své WiFi síti. Knihovna WiFiManager řešila poskytnutí captive portálu, vytvořením HTTP serveru a WiFi přístupového bodu, dále pak validovala zadané údaje, připojila modul k lokální WiFi síti, a uložila přístupové jméno a heslo do paměti mikropočítače.

Stejné řešení pro modul ESP32 ovšem nešlo využít, z toho důvodu, že v době implementace nebyla pro modul ESP32 k dispozici žádná knihovna podobná svou funkcí WiFiManageru, uživatelé by tedy byli odkázáni na manuální zadávání WiFi přístupových údajů. U verze pro

modul ESP32 byla tedy místo captive portálu použita funkce SmartConfig, která umožňuje „odvysílání“ přístupových údajů pomocí mobilní aplikace. Princip funkce je následující: inicializací knihovny SmartConfig začne zařízení naslouchat na broadcastové UDP zprávy, uživatel si stáhne mobilní aplikaci SmartConfig a vloží do ní přihlašovací údaje ke své WiFi síti, aplikace odešle zašifrované údaje UDP broadcastem, zařízení je přijme a pokusí se připojit k WiFi síti, v případě úspěchu odešle multicastem potvrzující informaci uživateli [51].

Vzhledem k tomu, že SmartConfig neřeší perzistentní uložení přijatých údajů, uživatel by musel tento postup opakovat při každém zapnutí zařízení, implementoval jsem tedy do knihovny EspHubLib funkci, která knihovnu SmartConfig inicializuje a získané přihlašovací údaje uloží do flash paměti. Uživatel tak pouze ve svém programu zavolá metodu `handleWifiConnection` a při prvním připojení zadá v mobilní aplikaci přihlašovací údaje k bezdrátové síti. Pokud by se zařízení nepodařilo po restartu zařízení připojit k bezdrátové síti, vyzve uživatele k použití aplikace a opětovnému zadání přihlašovacích údajů. Metoda `handleWifiConnection` je prozatím k dispozici pouze ve verzi knihovny pro ESP32, u verze pro ESP8266 nebyla metoda implementována, protože v době vzniku nebyla funkce SmartConfig dostatečně stabilní.

7.3 Vstupní zařízení

Vstupní zařízení, v systému nazývané také jako input ability, je obecně libovolný senzor poskytující data. V současnosti systém dokáže vizualizovat pouze data, která jsou v číselném, nebo textovém formátu, jiné typy dat, jako například obraz, nebo binární data, nebudou uživateli korektně zobrazeny ve webovém rozhraní, což ovšem neznamená že je uživatel systému nemůže zaslat. EspHubServer do své databáze dokáže uložit jakýkoliv záznam ve formátu JSON o velikosti maximálně 1024 znaků, podmínkou je ovšem aby daná abilita byla pro dané zařízení registrována.

Registraci ability uživatel provádí při inicializaci knihovny pomocí metody `setAbilities`, které předá seznam názvů poskytovaných abilit. Typ dané ability (vstupní, nebo výstupní), zobrazovanou jednotu a jméno si pak uživatel může upravit ve webovém rozhraní serveru při registraci zařízení.

7.4 Výstupní zařízení

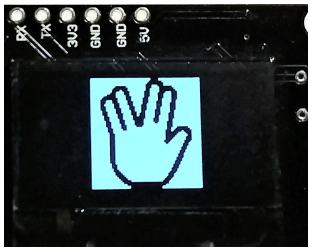
Výstupní zařízení (output ability) mohou být tlačítka, spínače, nebo displeje. Tlačítka na zařízení vyvolávají jednorázovou bezstavovou akci, například odečtení stavu senzoru. Naproti tomu spínač dokáže udržovat v systému stav vypnuto/zapnuto, tento ovládací prvek se hodí například pro spínání motorku, uživatel ve webovém portálu vidí, zda je zařízení aktivní, nebo ne. Zařízení může, pomocí zprávy, zpětně indikovat zdali zařízení je nebo není vypnuté.

Pro registrované výstupní akce si uživatel ve svém programu musí implementovat vhodné chování, což lze učinit pomocí registrace callback funkce metodou `setCallback`. Callback funkci může uživatel registrovat pouze jednu, pokud má tedy více výstupních zařízení musí v callback

funkci vytvořit rozhodovací logiku, která určí na jakém zařízení bude akce provedena. Callback funkce obdrží při zavolání 3 argumenty, téma zprávy, obsah zprávy a délku zprávy, obsahem zprávy je JSON, který zahrnuje kromě příkazu k aktivaci, nebo deaktivaci i název dotyčné ability.

Speciálním případem jsou výstupní zařízení typu display, v tuto chvíli jsou podporovány pouze monochromatické OLED displeje s řadičem SSD1306 a částečně barevné LCD displeje ILI9341. Vzhledem k rozdílným vlastnostem různých displejů nebylo možné implementovat univerzální rozhraní pro přenos obrazu. Displeje SSD1306 jsou monochromatické, mohou tedy využívat úsporné kódování obrazu popsané v kapitole 8.1, snímek obrazovky lze pak přenést pomocí jedné MQTT zprávy. Na obrázku 25 je ukázka bitmapy zobrazené na displeji SSD1306. Displeje ILI9341 mají rozlišení 320x240 pixelů a podporu barev. U tohoto displeje nelze jednotlivé snímky obrazovky, kvůli velikosti, přenést jednou MQTT zprávou, protože paměť modulu ESP8266 není dostatečně velká aby jej pojala, snímek se tedy musí přenášet po řádcích.

Z těchto důvodů musí být na serveru implementována speciální metoda ke zpracování obrazu pro každý konkrétní model displeje. Rovněž v programu zařízení musí být uživatelem implementována samostatná obsluha přijaté obrazové zprávy. Z důvodu snadnější obsluhy nejsou obrazová data šířena standardním tématem pro zprávy, ale používají vlastní téma `esp_hub/device/ID/display`.



Obrázek 25: Ukázka zobrazení bitmapy na displeji SSD1306

7.5 Implementace

Jak již bylo zmíněno knihovna je vytvořena pro platformu Arduino a používá standardní knihovny pro práci s moduly ESP8266 a ESP32. Jedná se například o knihovny WiFi, WiFiUdp, EEPROM, která zajišťuje přístup k flash paměti, nebo `ESPmDNS` pro funkci SmartConfig. Pro vytváření, dekódování a validaci JSON zpráv se používá knihovna ArduinoJson vytvořená vývojářem Benoîtem Blanchonem.

Komunikaci s MQTT obstarává knihovna Pubsubclient udržovaná Nickem O'Learym, která byla pro potřeby tohoto projektu mírně upravena. Knihovna dokáže ve výchozím stavu zpracovat zprávy o velikosti nejvýše 128 B, což dostačuje pro odesílání ze senzorů, ale pro účely přenosu obrazových dat bylo nutné tuto hodnotu navýšit na 512 B.

8 Knihovna EspHubUnilib

Knihovna EspHubUnilib byla vytvořena, jako částečná reimplementace knihovny EspHubLib ovšem pro zařízení s operačním systémem podporující jazyk Python, tedy Linux, MacOS, Windows a další. Přestože, je možné knihovnu spustit téměř na jakémkoliv zařízení s podporou Pythonu, většina funkcionalit je odladěna pro operační systém Linux. Knihovna EspHubUnilib se dokáže připojit jako standardní zařízení k EspHubServeru, zaregistrovat své výstupní vlastnosti (ability) a odesílat data. Prozatím ovšem chybí zpětný komunikační kanál ze serveru směrem na zařízení, na druhou stranu knihovna EspHubUnilib disponuje doplňkovým modulem ImageTransmitter, díky kterému dokáže odesílat data na jiná zařízení s připojeným displejem. Stejně jako EspHubServer jde i EspHubUnilib rozdělit na více funkčních komponent, v tomto případě se jedná o samotnou komponentu klientského zařízení EspHubUnilib a o již zmíněnou komponentu ImageTransmitter.

Oba nástroje uživatel ovládá pouze pomocí rozhraní příkazové řádky vytvořeného, stejně jako u skriptu z knihovny EspHubServer, za pomoci knihovny Click. Většina nastavení nástroje EspHubUnilib je uložena v konfiguračním souboru, zde jsou deklarovány i ability, které zařízení poskytuje.

Výstupní ability uživatel definuje pomocí skriptů v jazyce Python, skript musí obsahovat funkci, která nepřijímá žádný argument a vrací jednu serializovatelnou hodnotu, tedy číslo, řetězec znaků a podobně. V konfiguračním soubor poté deklaruje název ability, časový interval odesílání dat, skript s funkcí a konkrétní název funkce, která poskytuje data, pokud nezadá uživatel žádnou funkci použije se funkce `run`. Ve výpisu 1 je ukázka deklarace ability, která každách 60 s zasílá informace o počasí. EspHubUnilib má implementovaný plánovač úloh, který je kopií plánovače úloh z EspHubServeru, může tedy zpracovávat více úloh paralelně. Tento plánovač následně spouští v definovaných intervalech kód jednotlivých abilit a výsledky odesílá na server.

```
[weather_karvina]
interval = 60
function = weather_karvina
script_path = ability_functions.py
```

Výpis 1: Ukázka deklarace ability v konfiguračním souboru knihovny EspHubUnilib.

8.1 ImageTransmitter

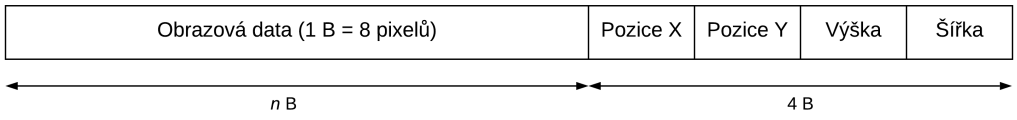
Komponenta ImageTransmitter byla vytvořena za účelem přenášení obrazových dat na displej připojený k ESP zařízení, její účel je tedy podobný obrazovému generátoru z knihovny EspHubServer, ale možnosti jsou daleko širší. ImigeTransmitter umožňuje přenášet obraz i na zařízení, která neparticipují v systému EspHub a podporuje jak protokol MQTT, tak UDP. Obrazová data lze na zařízení odesílat těmito metodami:

Jednotlivé snímky Pomocí příkazu `send-image` lze odeslat jeden konkrétní obrázek předaný jako parametr.

Více snímků Příkazu `send-images` se předá jako argument složka s více snímky, ImageTransmitter následně tyto snímky začne odesílat v nekonečné smyčce s nastavitelnou snímkovací frekvencí, na displeji se takový proud snímků bude jevit jako video.

Pojmenovaná roura Příkazem `pipe-interface` vytvoří ImageTransmitter v hostitelském systému pojmenovanou rouru, která přímá data na stdin rozhraní, transformuje vstup do správného formátu a odesílá jej na zařízení. Uživatel tak může spustit jiný program jehož výstupem budou obrazová data a ten v reálném čase odesílat na zařízení.

Je nutné zdůraznit, že tato komponenta je určená především k přenosu obrazu na jednoduché monochromatické displeje s 1 bitovou hloubkou obrazu, tomu je uzpůsoben i formát ve kterém jsou data přenášena. Všechny zmíněné metody pro přenos obrazu na svém vstupu očekávají monochromatické snímky s 1 bitovou hloubkou obrazu, pokud uživatel nemá data v tomto formátu, může pomocí přepínače `normalize` vynutit převod na tento formát. Podoba výsledného formátu dat je zachycena na obrázku 26, prvních n bytů představuje samotná obrazová data, přičemž do každého bytu je zakódováno 8 pixelů, následují 4 byty informující zařízení o počáteční pozici obrazu na ose x a y, výšce obrazu a šířce obrazu.



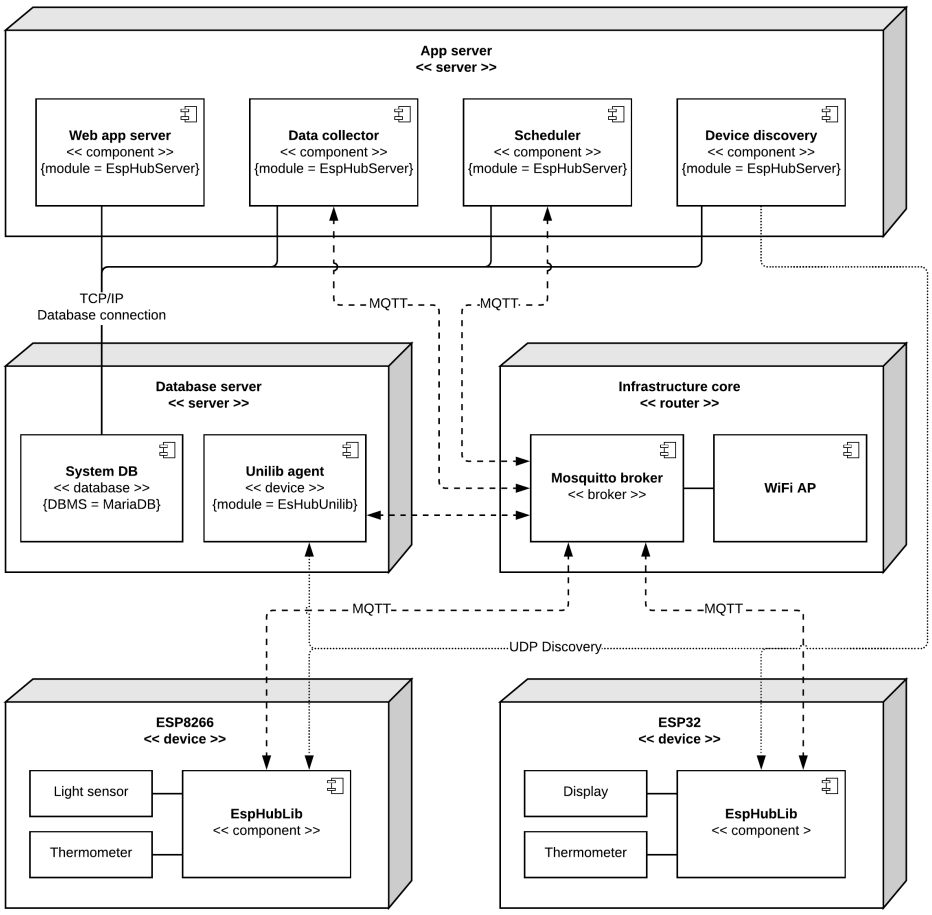
Obrázek 26: Struktura zprávy s obrazovými daty.

Aby uživatel mohl přenést snímek na zařízení přes MQTT musí znát ID zařízení, to totiž tvoří identifikátor tématu, na kterém zařízení naslouchá. Pro odeslání přes UDP je nutné znát IP adresu zařízení. Vzhledem k tomu že tyto údaje nemusí být uživateli vždy k dispozici je ImageTransmitter vybaven funkcemi pro překlad jména zařízení na příslušné ID, nebo IP adresu, tyto funkce lze ovšem využít pouze pokud je zařízení registrováno v systému EspHub, pro překlad jména zařízení je pak na serveru používán modul MQTT API.

9 Případový scénář

V této kapitole jsem se pokusil namodelovat jeden z možných případů využití systému EspHub. Systém bude tvořen dvěma servery, první z nich **App server** bude hostit všechny komponenty EspHubServeru, bude tedy poskytovat webové rozhraní, sbírat data, zajišťovat běh plánovače úloh a rozesílat discovery zprávy. Druhý server, označený jako **Database server**, bude určen pro hostování databáze a bude v něm běžet jedna instance EspHubUnilib, která bude pravidelně (každých 30 s) posílat informace o využití disku.

Dále budou k systému připojeny dvě zařízení, modul ESP8266 a ESP32. ESP8266 bude mít k sobě, přes I2C sběrnici, připojen senzor osvětlení BH1750FVI a odesílat serveru, jednou za minutu, data o aktuální intenzitě osvětlení. Modul ESP32 bude mít, rovněž přes I2C sběrnici, připojen OLED displej SSD1306 a bude zobrazovat aktuální intenzitu osvětlení. Obraz pro displej bude připravovat a odesílat komponenta Scheduler, která bude obsah displeje aktualizovat přibližně jednou za minutu. Oba moduly budou, každých 15 s, odesílat data o teplotě, které si uživatel může prohlédnout ve webovém rozhraní.



Obrázek 27: Diagram nasazení demonstračního systému EspHub.

Spojení mezi zařízeními a serverem bude obstarávat **Infrastructure core** router, ten kromě běžného směrování a WiFi připojení bude hostovat i MQTT Mosquitto broker.

Pro uchovávání dat bude použit databázový systém MySql, který je provozován na Database serveru. V ukázkovém nasazení bude aktivní také funkce automatického objevování modulů. Za rozesílání discovery zpráv bude zodpovědný modul Device discovery běžící na App serveru.

Na obrázku 27 je znázorněno schématické propojení jednotlivých částí systému.

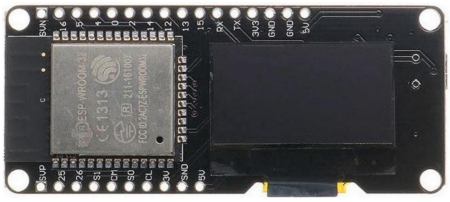
9.1 Použitý hardware

Jako zástupce rodiny modulů ESP32 jsem zvolil vývojovou desku Wemos Lolin s OLED displejem. Přestože většina zdrojů tuto desku nazává Označení této desky je poněkud zavádějící, přestože ji většina prodejců nazývá Wemos Lolin ESP32, nemá nic společného se stejnojmennou společností Wemos, jedná se spíše o neznačkový klon desky D-duino-32 V3 od výrobce DSTIKE [52].

Deska je osazena modulem ESP-WROOM-32 výrobce Espressif System, vybaveným 4 MB externí SPI flash paměti. Modul obsahuje také integrovaný Hallův senzor a teploměr, který slouží pro odesílání teplotních dat do systému. Specifikem vývojové desky je integrovaný monochromatický OLED displej s řadičem SSD1306, který je připojen k desce přes rozhraní I2C. Displej operuje na I2C adrese `0x3c` a jeho vstupy SDA a SDC jsou napevno připojeny ke výstupům mikropočítače 5 a 4.

Druhou vývojovou deskou je NodeMCU V1 s ESP8266-12E a externí SPI flash pamětí velikosti 4 MB. K počítači jsem přes rozhraní I2C připojil senzor ambientního osvětlení BH1750FVI, která dokáže měřit intenzitu osvětlení v rozsahu 1-65535 lx. Druhým senzorem je teploměr Dallas DS18B20 připojený rovněž přes I2C rozhraní. Teploměr zvládá měřit teplotu v rozsahu -55 až 125 °C s přesností na 0,5 °C.

Oba mikropočítače jsou vybavený micro USB rozhraním pro ladění a programování. Podoba modulů je na obrázcích 28 a 29.



Obrázek 28: Počítač NodeMCU s ESP8266 [53]. Obrázek 29: Počítač Wemos lolin s ESP32 [54]

9.2 Použitý software

Kromě knihovny EspHubLib a přidružených knihoven používají moduly ESP ještě několik specifických knihoven pro ovládání připojených periferií. Modul ESP8266 používá knihovu OneWire pro komunikaci po I2C sběrnici. Pro usnadnění s připojenými senzory jsou použil Arduino knihovny BH1750 a DallasTemperature. Modul ESP32 má teploměr integrovaný, není tedy zapotřebí žádná obslužná knihovna, použita je pouze knihovna pro ovládání displeje SSD1306Wire [55]. Pro řízení tohoto typu displeje je pro Arduino k dispozici více knihoven, nejčastěji používané jsou například Adafruit_SSD1306, nebo u8glib, ani jedna z nich ovšem nebyla v době implementace plně kompatibilní s použitým modulem.

Oba použité servery jsou poháněny systémem Debian 8. Router Infrastructure core pohání systém Linux turris založený na OpenWrt. Na routeru běží MQTT broker Mosquitto 4.1.9. Na všech těchto zařízeních je nainstalován Python ve verzi 3.5. Jako DBMS se používá MySql Server ve verzi 14.14.

Vzhledem k tomu, že procesy EspHubServer a EspHubUnilib nemají nativní podporu pro běh na pozadí, je nutné použít nástroje operačního systému, například Supervisor, nebo nástroj nohup, který byl použit v tomto případě.

9.3 Testování obecných parametrů

Případový scénář jsem zprovoznil v domácím nasazení a provedl několik orientačních testů. Sběr dat probíhal po dobu 3 dnů a bylo nasbíráno bezmála 50 tisíc datových záznamů.

Bezdrátová konektivita V tomto testu jsem pozoroval chování při různých úrovních kvality bezdrátové konektivity. Oba zařízení jsem přesouval do různé vzdálenosti od WiFi přístupového bodu a sledoval jejich úroveň RSSI a schopnost odesílat a přijímat data. Test jsem provedl v běžné domovní zástavbě, první měření proběhlo ve vzdálenosti 2 metrů od přístupového bodu, bez překážek v prostoru. Druhé měření ve vzdálenosti 7 metrů, s jednou cihlovou stěnou v cestě signálu. Poslední měření jsem prováděl přibližně ve vzdálenosti 12 metrů od přístupového bodu, s 1 cihlovou stěnou a stropem o tloušťce přibližně 0,5 m.

V tabulce 5 je zachycena závislost síly přijímaného signálu (RSSI) na umístění vůči WiFi přístupovému bodu. Hodnoty signálu v nejvzdálenějším měřeném umístění představují praktické maximum pro nasazení modulů. Při signálu o síle nižší než -85 dB dochází k občasnému vypadávání bezdrátového spojení. Po všech pozorovaných případech ztráty konektivity se zařízením opět připojilo a navázalo komunikaci se serverem, a to vždy nejvýše do 30 s po obnovení dostatečné úrovně signálu. Ani při horší úrovni signálu nebyly zaznamenány ztráty odeslaných hodnot, nebo poruchy v přenosu obrazu.

Zařízení	Situace 1 (2 m)	Situace 2 (7 m)	Situace 3 (12 m)
ESP8266	-41 dB	-66 dB	-84 dB
ESP32	-39 dB	-61 dB	-82 dB

Tabulka 5: Závislost síly signálu (RSSI) na vzdálenosti zařízení od WiFi přístupového bodu.

Rychlost připojení Dalším testovaným parametrem byla rychlost připojení modulu k serveru. Měřen byl časový úsek mezi zapnutím modulu a prvním odeslání telemetrické zprávy serveru. Pro zjištění časového úseku jsem použil ladící výpisy odesílané na sériové rozhraní. Celkem jsem tímto způsobem provedl 40 měření na obou modulech.

V tabulce 6 jsou zachyceny minimální, maximální a průměrné doby od zapnutí po odeslání první telemetrie. Z měření vyplývá, že procedura připojování je u modulu ESP8266 o něco pomalejší. Tyto rozdíly lze pravděpodobně připsat nižšímu výkonu modulu ESP8266, oproti ESP32, a také nutností inicializace knihovny WiFiManager, která obstarává přihlášení k bezdrátové síti.

Zařízení	Minimální doba	Maximální doba	Průměrná doba
ESP8266	3,7 s	4,1 s	3,8 s
ESP32	1,9 s	2,4 s	2,2 s

Tabulka 6: Naměřené doby mezi zapnutím modulu a odesláním první telemetrické zprávy.

Přenos obrazu V tomto scénáři jsem zjišťoval, kolik snímků dokáže server přenést na displeje zařízení. Pro testování jsem v tomto případě nepoužil standardní metodu generování snímků, kterou v systému obstarává komponenta Scheduler, ale komponentu ImageTransmitter z knihovnu EspHubUnilib. Důvodem pro toto rozhodnutí byla hardwarová náročnost standardního procesu pro generování snímků pro displeje, popsaného v kapitole 8.1. Standardní proces dokáže vytvořit jeden snímek přibližně za 15 s, což nedostačuje pro relevantní otestování. Snímky je možné generovat i paralelně, ale náročnost na hardware je vysoká a mohla by způsobit nepravidelné odesílání.

Za účelem zátěžového testu jsem si vytvořil sadu 10 snímků, které jsem pomocí komponenty ImageTransmitter odesílal jako konstantní proud snímků, který se na displeji zařízení jevil jako video. Funkci `send-images`, kterou jsem použil, je možné pomocí parametru `--frame-rate` nastavit kolik snímků za sekundu má odeslat.

Maximální frekvenci, které jsem byl schopen dosáhnout bylo 18 snímků za sekundu. Při vyšších rychlostech odesílání se projevuje efekt „dobíhání“, MQTT klient nestačí zprávy odbavovat a ty se hromadí ve frontě brokeru, zařízení tak snímky nezobrazuje v reálném čase, ale až několik sekund po odeslání. Po delší době nepřetržitého odesílání dojde k zahlcení fronty brokeru a ztrátě snímků.

Pro srovnání jsem stejný test provedl ještě jednou, ale místo protokolu MQTT jsem použil protokol UDP, který komponenta ImageTransmitter podporuje. Maximální zobrazovací frek-

vence v tomto případě přesahovala 60 snímků za sekundu, což je více, než dokáže použitý OLED displej zobrazit. Důvodem značně vyššího výkonu protokolu UDP je menší velikost zpráv, ale především minimum operací prováděných nad přijatými daty. Zařízení nemusí obsluhovat spojení s MQTT brokerem a může zachycený paket okamžitě odeslat na displej.

9.4 Testování propustnosti sítě senzorů

Ukázkový scénář reprezentuje malou domácí síť senzorů, množství MQTT zpráv, které projde systémem je v řádu desítek za minutu. V tomto testovacím scénáři jsem se pokusil zjistit maximální limity sytému. Za tímto účelem jsem, pomocí knihovny EspHubUnilib, nasimuloval 10 nových zařízení s proměnlivou frekvencí odesílání naměřených dat. Pomocí těchto virtuálních zařízení jsem dokázal generovat umělou zátěž na komponentu pro sběr MQTT zpráv. Odesílaná data měla podobu náhodných celočíselných hodnot.

Z tabulky 7 je patrné, že systém je schopen zpracovat přibližně 1000 zpráv za minutu. Při vyšší frekvenci odesílání zpráv dochází k tomu, že komponenta Data collector nestíhá z brokeru odebírat zprávy a ty se hromadí ve frontě brokeru. Po překročení určitého počtu zpráv dojde k přeplnění fronty a některé zprávy jsou zahozeny. Opakuje se tak efekt „dobíhání“ zpráv, podobně jako u displejů.

Odesláno zpráv	200	400	600	800	1000	1200	1400	1600	1800	2000
Zpracováno správ	200	400	600	800	1000	1118	1132	1137	1129	1131

Tabulka 7: Poměr počtu odeslaných zpráv za minutu vůči zpracovaným zprávám za minutu.

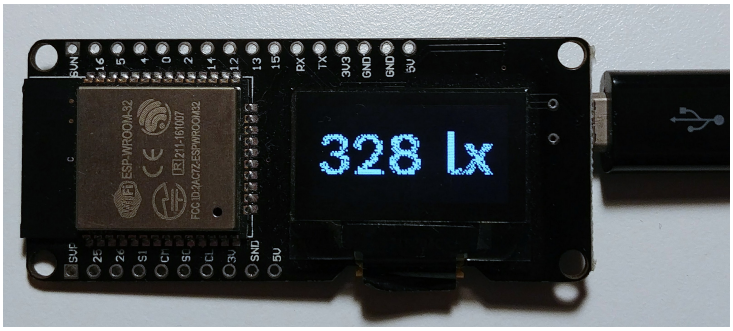
Proces Data collectoru neimplementuje žádnou frontu zpráv, ve vlákně pro zpracování přijatých zpráv (jejich validaci a uložení do databáze) probíhá i obsluha spojení s MQTT brokerem. Výsledkem toho je špatná efektivita procesu pro sběr dat, zpracování každé zprávy trvá přibližně 900 μs, což koresponduje s naměřeným množstvím zpracovaných dat.

Tato část aplikace by se dala výrazně zefektivnit implementací návrhového vzoru Producer and Consumer. Příchozí zprávy by byly hlavním vláknem, bez jakéhokoliv zpracování, vkládány do fronty zpráv. Systém by zároveň udržoval n vláken, nebo procesů, které by z této fronty paralelně vyzvedávaly zprávy, validovaly je a následně ukládaly do databáze. Maximální kapacita přijatých zpráv by se pak dala škálovat zvolením správného množství procesů n.

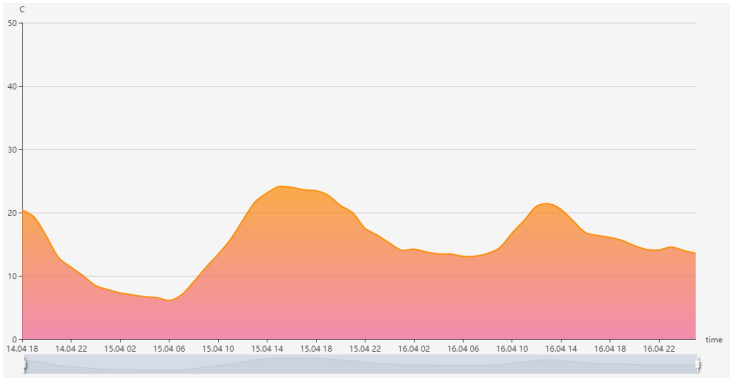
9.5 Shrnutí

Případový scénář demonstruje nasazení systému EspHub v domácím prostředí. Provedené testy ověřili správné fungování základních mechanizmů systému a poukázaly na možné limitace. Tou je především datová propustnost, která se pohybuje okolo 1000 zpracovaných zpráv za minutu, což ovšem pro domácí prostředí dostačuje. Omezením je také WiFi připojení, síla signálu se výrazně mění v závislosti na okolním prostředí a v prostoru s mnoha překážkami mohou být senzory nasazeny maximálně do vzdálenosti několika desítek metrů.

Na obrázku 30 je ukázka naměřené hodnoty osvětlení, zobrazené na displeji vývojové desky Wemos lolin. Obrázek 31 znázorňuje graf měnící se teploty zaznamenaný mikropočítačem NodeMCU.



Obrázek 30: Naměřená hodnota osvětlení zobrazená na displeji.



Obrázek 31: Graf změny teploty ve sledovaném časovém úseku.

10 Závěr

Společně s rozvíjejícím se trendem IoT vzniklo v posledních letech několik volně dostupných nástrojů zabývajících se komplexní správou IoT systémů, v této práci jsem vyzkoušel a popsal 6 z nich. Zaměřil jsem se na ty nejpoužívanější a nejzajímavější, konkrétně: Domoticz, ESPEasy, Blynk, OpenHAB, Home Assistant a ThingsBoard. Všechny analyzované systémy, pro řízení IoT, sdílí podobné rysy, pokusil jsem se tedy stanovit seznam základních požadavků na tyto systémy. Optimální systém by měl disponovat alespoň těmito funkcemi: obousměrná komunikace, zobrazení stavu zařízení, vizualizace dat, automatické navázání spojení a plánování úloh.

Nejpodstatnější částí internetu věcí jsou použité mikropočítačové moduly, ty tvoří kostru systému, sbírají data, mohou manipulovat s periferiemi a zajišťují komunikaci s okolním světem. V této práci jsem se zaměřil na moduly ESP8266 a ESP32, ty svou nízkou cenou a všestrannými schopnostmi akcelerovaly zájem o systémy domácí automatizace. ESP8266 je malý mikropočítač vybavený bezdrátovou WiFi konektivitou, disponuje také 17 programovatelnými GPIO, díky čemuž jej lze použít pro ovládání širokého spektra periferií. ESP32 je nástupcem předchozího modulu a vylepšuje jej ve všech ohledech, přidává například podporu Bluetooth, více GPIO, větší množství rozšiřitelných sběrnic a další.

Prozkoumal jsem většinu aktuálně dostupných nástrojů k vývoji aplikací pro tyto moduly, nejvíce pozornosti jsem však věnoval vývojovým frameworkům kompatibilních s platformou Arduino. Ta totiž přináší řadu výhod, uživatel má k dispozici velké množství rozšiřitelných knihoven, pohodlné vývojové nástroje a širokou podporu komunity. Arduino SDK je také v současnosti nejpoužívanějším nástrojem pro vývoj. V případě modulu ESP8266 jej udržuje aktivní komunita vývojářů, u ESP32 se o vývoj stará sám výrobce Espressif System. Ten vytváří také vývojové nástroje založené na operačním systému FreeRTOS, kromě nich existuje ještě řada komunitních platforem na bázi programovacích jazyků Lua, Python, JavaScript a dalších.

Na základě analýzy existujících řešení jsem navrhl systém pro řízení modulů ESP, který jsem nazval EspHub. Systém je postaven na architektuře klientů a serveru, kteří spolu komunikují pomocí protokolu MQTT. Tento protokol jsem zvolil, protože byl navržen pro použití v IoT a mnoho jiných systémů jej také používá. MQTT umožňuje spolehlivou komunikaci v prostředí s horší kvalitou konektivity, je nenáročný na hardwarové prostředky a obsahuje mechanizmy pro oddělení zpráv dle jejich tématu. Centrálním komunikačním bodem je MQTT broker, skrze který klienti i server odesílají zprávy.

EspHub se skládá s 3 nezávislých částí: serverové aplikace (EspHubServer), klientské aplikace pro moduly ESP (EspHubLib) a klientské aplikace pro operační systémy s podporou jazyka Python (EspHubUnilib). Server je vytvořen v jazyce Python, spravuje dostupná zařízení a provádí sběr dat ze zařízení, jejich uložení do databáze, vizualizaci uživateli skrze webové rozhraní a odesílání dat zpět na zařízení. Jednou z inovativních funkcí je takzvaný discovery protokol, díky němuž jsou zařízení schopna automaticky nalézt lokální server. Server následně notifikuje uživatele a vyzve ho k validaci zařízení.

Server je schopen provádět jednorázové i naplánované úlohy odesílání dat na zařízení, jedná se zejména o činnosti ve smyslu provedení akce na zařízení, nebo aktivace/deaktivace periferie. Nekonvenční funkcí je ovšem možnost ovládat displeje připojené k zařízení. Uživatel si ve webovém rozhraní definuje obsah zobrazený na displeji pomocí HTML kódu, podobně jako webovou stránku, server následně zajistí převod tohoto obsahu na snímek, upraví jej pro potřeby konkrétního zařízení a odešle jej protokolem MQTT.

Pro plnohodnotnou komunikaci zařízení ESP se serverem jsem vytvořil knihovnu EspHubLib. Knihovna je implementována pomocí vývojových nástrojů pro platformu Arduino. Uživatel pomocí knihovny oznámí, jaké funkce bude zařízení poskytovat, zaregistruje se k serveru a pomocí poskytovaných metod může odesílat serveru data. K dispozici jsou také metody pro příjem zpráv ze serveru. Knihovna v pravidelných intervalech notifikuje server o stavu zařízení, síle signálu a adrese, díky těmto telemetrickým zprávám má server přehled o aktuálně dostupných zařízeních.

Poslední vytvořenou komponentou je EspHubUnilib, ta víceméně simuluje chování zařízení ESP, kromě toho však přináší i funkci pro odesílání obrazu na displeje připojené k jiným zařízením.

V závěru práce jsem nastínil jeden ukázkový scénář nasazení systému v domácích podmínkách. Prototypové nasazení systému obsahovalo řídící infrastrukturu a dva mikropočítače ESP. První z nich vystupoval v roli pasivního senzoru osvětlení a teploty, druhý dokázal zobrazovat, na připojeném displeji, přijatá data o intenzitě světla.

Nad ukázkovým systémem jsem provedl několik testů týkajících se zejména kvality připojení k bezdrátové síti a propustnosti dat. Systém vyhověl požadovaným parametrům, ale vyvstaly zde i příležitosti pro případná vylepšení do budoucnosti, zejména v oblasti kapacity zpracovávaných zpráv.

Možnosti vytvořeného systému nemohou plně konkurovat schopnostem velkých existujících řešení, i přesto jsem se od počátku snažil navrhnout systém tak, aby byl snadno rozšiřitelný a otevřený pro možná vylepšení. Kompletní řešení jsem dal tedy volně k dispozici na portálu GitHub, pod svobodnou licencí MIT [56].

Literatura

[1] Hernando Barragán. *The Untold History of Arduino* [online]. [cit. 20.04.2018]. Dostupné z: `https://arduinohistory.github.io`

[2] *ESP-12E: ESP8266 Serial Port WIFI Wireless Transceiver Module for Arduino* [online]. [cit. 20.04.2018]. Dostupné z: `www.botnroll.com/en/xbee-rf/2149-esp-12e-esp8266-serial-port-wifi-wireless-transceiver-module-for-arduino-.html`

[3] Petr Stehlík. *ESP32 je tu. Co přinese nástupce ESP8266?* [online]. Copyright © Root.cz [cit. 20.04.2018]. Dostupné z: `www.root.cz/clanky/esp32-je-tu-co-prinese-nastupce-esp8266/`

[4] *UUID objects according to RFC 4122* [online]. Python Software Foundation [cit. 20.04.2018]. Dostupné z: `https://docs.python.org/3.2/library/uuid.html?highlight=uuid`

[5] *A Universally Unique IDentifier (UUID) URN Namespace* [online]. Copyright © The Internet Society [cit. 20.04.2018]. Dostupné z: `https://tools.ietf.org/html/rfc4122.html`

[6] *Blockly* [online]. Domoticz [cit. 2018-04-26]. Dostupné z: `http://www.domoticz.com/wiki/Blockly`

[7] *Hardware* [online]. Domoticz [cit. 2018-04-26]. Dostupné z: `http://www.domoticz.com/wiki/Hardware`

[8] *Integrations and Protocols* [online]. Domoticz [cit. 2018-04-26]. Dostupné z: `http://www.domoticz.com/wiki/Integrations_and_Protocols#Protocols`

[9] *OpenHAB 2 - Empowering the Smart Home*. OpenHAB 2 Documentation [online]. OpenHAB Foundation [cit. 27.04.2018]. Dostupné z: `https://docs.openhab.org/introduction.html`

[10] *Home Assistant - Hass.io* [online]. Home Assistant [cit. 27.04.2018]. Dostupné z: `https://www.home-assistant.io/hassio/`

[11] *Home Assistant - Components* [online]. Home Assistant [cit. 27.04.2018]. Dostupné z: `https://www.home-assistant.io/components`

[12] *Home Assistant - Addons* [online]. Home Assistant [cit. 27.04.2018]. Dostupné z: `https://www.home-assistant.io/addons/`

[13] *IFTTT*. IFTTT helps your apps and devices work together [online]. IFTTT Inc. [cit. 27.04.2018]. Dostupné z: https://ifttt.com

[14] *Report the temperature with ESP8266 to MQTT*. Home Assistant [online]. Home Assistant [cit. 27.04.2018]. Dostupné z: https://www.home-assistant.io/blog/2015/10/11/measure-temperature-with-esp8266-and-report-to-mqtt/

[15] *ThingsBoard*. ThingsBoard - Open-source IoT Platform [online]. Copyright © 2018 The ThingsBoard Authors [cit. 27.04.2018]. Dostupné z: https://thingsboard.io/products/

[16] *Espressif IoT Development Framework*. Official development framework for ESP32 [online]. Espressif System [cit. 27.04.2018]. Dostupné z: https://github.com/espressif/esp-idf

[17] *Arduino core for the ESP32* [online]. Espressif System [cit. 27.04.2018]. Dostupné z: https://github.com/espressif/arduino-esp32

[18] *Getting Started Guide - ESP8266*. Espressif Systems - Wi-Fi and Bluetooth chipsets and solutions [online]. Espressif System [cit. 27.04.2018]. Dostupné z: https://www.espressif.com/en/support/explore/get-started/esp8266/getting-started-guide

[19] Neil Kolban. *Kolban's Book on the ESP32 & ESP8266* [online]. Leanpub, Copyright © 2016 [cit. 27.04.2018]. Dostupné z: https://leanpub.com/ESP8266_ESP32

[20] Paul Sokolovsky. *Integrated SDK for ESP8266/ESP8285 chips* [online]. [cit. 27.04.2018]. Dostupné z: https://github.com/pfalcon/esp-open-sdk

[21] Max Filippov. *GCC for Xtensa* [online]. [cit. 27.04.2018]. Dostupné z: https://github.com/jcmvbkbc/gcc-xtensa

[22] Mikhail Grigorev. *Unofficial Development Kit for Espressif ESP8266* [online]. [cit. 27.04.2018]. Dostupné z: https://github.com/CHERTS/esp8266-devkit

[23] *What is PlatformIO?* PlatformIO documentation [online]. PlatformIO © Copyright 2014 [cit. 27.04.2018]. Dostupné z: http://docs.platformio.org/en/latest/what-is-platformio.html

[24] Ivan Grokhotkov. *ESP8266 core for Arduino* [online]. ESP8266 Community Forum [cit. 27.04.2018]. Dostupné z: https://github.com/esp8266/Arduino

[25] Daniel Eichhorn, Marcel Stör. *NodeMCU custom builds* [online]. [cit. 27.04.2018]. Dostupné z: https://nodemcu-build.com/index.php

[26] Dave. *4 reasons I abandoned NodeMCU/Lua for ESP8266*. Internet of Home Things [online]. Internet of Home Things [cit. 27.04.2018]. Dostupné z: https://internetofhomethings.com/homethings/?p=424

[27] *Mongoose OS - reduce IoT firmware development time up to 90%* [online]. Cesanta [cit. 27.04.2018]. Dostupné z: https://mongoose-os.com

[28] *Building and Running MicroPython on the ESP8266*. Adafruit Learning System [online]. Adafruit ® [cit. 27.04.2018]. Dostupné z: https://learn.adafruit.com/building-and-running-micropython-on-the-esp8266/overview

[29] *Espruino - JavaScript for Microcontrollers* [online]. Pur3 Ltd. [cit. 27.04.2018]. Dostupné z: https://www.espruino.com/

[30] *Sming - Open Source framework for high efficiency native ESP8266 development* [online]. SmingHub [cit. 29.04.2018]. Dostupné z: https://github.com/SmingHub/Sming

[31] *Flashing Instructions - ESP8266 BASIC*. ESP8266 BASIC - ESP BASIC [online]. ESP8266BASIC [cit. 29.04.2018]. Dostupné z: https://www.esp8266basic.com/flashing-instructions.html

[32] Nicholas Wang. *Esp32/esp8266 Lua SDK* [online]. [cit. 29.04.2018]. Dostupné z: https://github.com/Nicholas3388/LuaNode

[33] Jaume Olivé Petrus. *Lua RTOS for ESP32* [online]. Whitecat [cit. 29.04.2018]. Dostupné z: https://github.com/whitecatboard/Lua-RTOS-ESP32

[34] *ESP8266EX Datashet* [online]. Espressif System Inc., 2018 [cit. 29.04.2018]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[35] *ESP32 Datashet* [online]. Espressif System Inc., 2018 [cit. 29.04.2018]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[36] *ESP8266 Module Family*. ESP8266 Support WIKI - Everything ESP8266 [online]. ESP8266 Community Forum [cit. 29.04.2018]. Dostupné z: https://www.esp8266.com/wiki/doku.php?id=esp8266-module-familyhttps://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

[37] *The Internet of Things with ESP32* [online]. [cit. 29.04.2018]. Dostupné z: http://esp32.net

[38] Obermaier Dominik. *SYS Topics · mqtt/mqtt.github.io Wiki* [online]. Mqtt.org [cit. 29.04.2018]. Dostupné z: https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics

[39] *Configuring message hubs* [online]. International Business Machines Corp. (IBM) [cit. 29.04.2018]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSCGGQ_1.2.0/com.ibm.ism.doc/Administering/ad00360_.html

[40] Andy Stanford-Clark, Hong Linh Truong. *MQTT For Sensor Networks (MQTT-SN).* Protocol specification [online].
International Business Machines Corp. (IBM) [cit. 29.04.2018]. Dostupné z:
http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[41] *SQLAlchemy library* [online]. SQLAlchemy [cit. 29.04.2018].
Dostupné z: https://www.sqlalchemy.org/library.html

[42] *SQLAlchemy dialects* [online]. SQLAlchemy [cit. 29.04.2018].
Dostupné z: http://docs.sqlalchemy.org/en/latest/dialects/index.html

[43] *MQTT Essentials Part 1: Introducing MQTT* [online].
HiveMQ, dc-square GmbH [cit. 29.04.2018]. Dostupné z:
https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt

[44] *MQTT Essentials Part 6: Quality of Service Levels* [online]. HiveMQ, dc-square GmbH [cit. 29.04.2018]. Dostupné z: https://www.hivemq.com/blog/
mqtt-essentials-part-6-mqtt-quality-of-service-levels

[45] Steve Cope. *Introduction to MQTT Security Mechanisms - Beginners Guide.* Home Networking, MQTT and Home IOT [online]. [cit. 29.04.2018]. Dostupné z:
http://www.steves-internet-guide.com/mqtt-security-mechanisms/

[46] Ajay Kulkarni. *What the heck is time-series data (and why do I need a time-series database)?* [online]. [cit. 29.04.2018]. Dostupné z: https://blog.timescale.com/
what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series\
-database-dcf3b1b18563

[47] *Time Series Analysis - Statistics Solutions* [online]. Copyright © Statistics Solutions 2018 [cit. 29.04.2018].
Dostupné z: http://www.statisticssolutions.com/time-series-analysis/

[48] *InfluxDB | The Time Series Database in the TICK Stack* [online]. Copyright © 2018 InfluxData, Inc. [cit. 29.04.2018].
Dostupné z: https://www.influxdata.com/time-series-platform/influxdb/

[49] Prof. Dr. Stefan Edlich. *List of NOSQL databases* [online]. [cit. 29.04.2018].
Dostupné z: http://nosql-database.org

[50] Armin Ronacher. *Click Documentation (6.0)* [online]. Copyright © Copyright 2017, Armin Ronacher. [cit. 29.04.2018]. Dostupné z: http://click.pocoo.org/6/

[51] *How to use SmartConfig on Arduino ESP32.* IoT Sharing [online].
IoT Sharing [cit. 29.04.2018]. Dostupné z:
http://www.iotsharing.com/2017/05/how-to-use-smartconfig-on-esp32.html

[52] John Hart. *ESP32 With Integrated OLED (WEMOS/Lolin) - Getting Started Arduino Style*. Instructables - How to make anything [online]. Copyright © 2018 Autodesk, Inc. [cit. 29.04.2018]. Dostupné z: `http://www.instructables.com/id/ESP32-With-Integrated-OLED-WEMOSLolin-Getting-Star/`

[53] Danish Malhotra. *Getting Started With NodeMCU V1.0 and Blynk App*. Instructables - How to make anything [online]. Copyright © 2018 Autodesk, Inc. [cit. 29.04.2018]. Dostupné z: `http://www.instructables.com/id/\Getting-Started-With-NodeMCU-V10-and-Blynk-App/`

[54] *Wemos Lolin ESP32 OLED Module For Arduino ESP32 OLED WiFi+Bluetooth Dual ESP-32 ESP-32S ESP8266 OLED Module With ESP8266 Expansion Plate* [online]. Newegg.com [cit. 29.04.2018]. Dostupné z: `https://www.newegg.com/Product/Product.aspx?Item=285-000J-005Y8`

[55] Daniel Eichhorn. *Esp8266-oled-ssd1306: Driver for the SSD1306 and SH1106 based 128x64 pixel OLED display running on the Arduino/ESP8266 platform* [online]. ThingPulse [cit. 29.04.2018]. Dostupné z: `https://github.com/ThingPulse/esp8266-oled-ssd1306`

[56] Tanasis Vlachopulos. *EspHub - Implementation of intelligent modules with ESP8266 and ESP32* [online]. [cit. 29.04.2018]. Dostupné z: `https://github.com/TanasVlachopulos/EspHub`

A Příloha DVD

DVD obsahuje repozitář se zdrojovými kódy k systému EspHub.